ISI Research Report
ISI/RR-88-201
June 1988

4

University of Southern California

AD-A196 792

Gregory G. Finn

Reducing the Vulnerability of Dynamic Computer Networks

INFORMATION SCIENCES INSTITUTE ISI
213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT This document is approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ISI/RR-88-201	5. MONITORING ORGANIZATION REPORT NUMBER(S) --------------
6a. NAME OF PERFORMING ORGANIZATION USC/Information Sciences Institute — 6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION --------------
6c. ADDRESS (*City, State, and ZIP Code*) 4676 Admiralty Way Marina del Rey, CA 90292	7b. ADDRESS (*City, State, and ZIP Code*) --------------
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Defense Advanced Research Projects Agency — 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA903-87-C-0719
8c. ADDRESS (*City, State, and ZIP Code*) 1400 Wilson Boulevard Arlington, VA 22209	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
--------------	--------------	--------------	--------------

11. TITLE *(Include Security Classification)*
Reducing the Vulnerability of Dynamic Computer Networks [Unclassified]

12. PERSONAL AUTHOR(S) Finn, Gregory G.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Research Report	FROM ______ TO ______	1988, June	79

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
09	02	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
computer networks, network architecture, network design, network vulnerability, networks, protocols

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Networks are becoming important in the day-to-day operations of business, the military, and government. As the use of networks grows, it is a wise precaution to assume that malicious attempts to sabotage a network will occur. Network operating software should not make the network susceptible to widespread failure if one router, or even several, deviate from acceptable behavior. Network software should be resistant to this manner of attack while preserving the desirable network attributes of flexibility and efficiency. This report points out that several commonly used routing procedures imply a vulnerability to attack, and presents a routing procedure that allows the development of operating software that is highly resistant to attack.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☑ UNCLASSIFIED/UNLIMITED ☑ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Sheila Coyazo Victor Brown	22b. TELEPHONE *(Include Area Code)* 213-822-1511 — 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

ISI Research Report
ISI/RR-88-201
June 1988

University of Southern California

Gregory G. Finn

Reducing the Vulnerability of Dynamic Computer Networks

INFORMATION SCIENCES INSTITUTE

213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

This research is supported by the Defense Advanced Research Projects Agency under Contract No. MDA903 87 C 0719. Views and conclusions contained in this report are the author's and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government, or any person or agency connected with them.

Acknowledgments

I thank Jon Postel for his support of this work, Walter Prue who reviewed it and, finally, Sheila Coyazo and Diane Hatch-Avis for correcting the author's obscure or incorrect prose.

1 Introduction

The issue of computer network privacy and security has traditionally focussed on the *end-to-end* connection; the primary concern has been to prevent the unintended exposure of data, while the internal security of the computer network itself has largely been ignored. Typically, network communications strategies have implicitly assumed that the network provides a reliable means of data transport. If that assumption is untrue, then end-to-end connections may be denied service. If denial of service is a rare and isolated event, this is usually not a significant problem. However, if denial of service is either frequent or potentially widespread when it occurs, then the ability of the network to transmit any time-critical information (secure or otherwise) must be called into question.

Methods of dynamic routing used by computer networks may incorporate elements in their design that allow widespread denial of service. In 1980 an event occurred that illustrates this problem: a malfunction in one of the ARPANET's routers shut down the entire network for several hours [Rosen-81-2]. Although this incident was accidental, this type of failure could also be caused by a malicious attack.

Some long-haul networks today assume that the user population is trustworthy. Positive assumptions concerning these users, their motives, their mutual cooperation, and so on, are unrealistic in a widespread commercial environment. A network provides a peculiar attraction to malicious individuals, many of whom are technically adept. Recent news stories concerning the crimes of so-called network 'crackers' substantiate this. In military or security applications, the network designer must expect an enemy to exploit any design weakness.

Certain individuals, by virtue of their position, will have access rights to network hardware, software, and specialized information concerning network operation. It is highly possible that at some time someone with access will be malicious. It is unrealistic to expect network hardware or software to be completely tamper-proof.

As long-haul computer networks become an accepted part of day-to-day commercial or military operations, they become important and even vital to those operations. A carefully constructed hardware and software system cannot protect itself indefinitely against internal failure or human intervention. A distributed network must not incorporate design elements that allow widespread network failure as the result of an attack at any single place or a breakdown in a single network component. Network software should be resistant to attack.

1.1 Preliminaries

The non-local computer network is composed of various computers connected to one another by communications channels. Such a network can be abstractly modeled as an undirected graph in which the computers that perform the routing are represented by the nodes of the graph, and the communications channels are represented by the links between nodes. Networks facilitate communication by exchanging *packets* of information over these links. Packets are sent from a *source* to a set of *destinations*. We differentiate between computers that primarily perform the packet-routing functions of the network (*routers*) and those upon which people perform tasks (*hosts*). Routers are the intermediaries of the network that transport packets between source and destination. This report does not concern itself with computer networks that have bus, ring, or star topologies.

A computer network provides data transport, which is accomplished through the cooperation of routers and communications channels. A network can be damaged in several ways: poor routes, dropped packets, severed connections, poor channels, partitioning, router overloading, or congestion. However, to a user or application, the damage will be manifest as a denial of service. This report is primarily concerned with the function of routers, and how that function can expose the network to seriously damaging attacks. Other aspects of network security, such as data integrity, are discussed only insofar as they are used to make a network more resistant to attack. This report does not specifically concern itself with the issues involved in the secure transmission of end-to-end user traffic.

1.1.2 Practical Routing Requirements and Their Implications

Routing is a central issue in the design of any distributed computer network. Routing procedures are necessary if a dynamic network is to provide predictable service. [1] The routing procedures of early long-haul networks, such as the ARPANET, were designed around the following three requirements:

(1) Reliable delivery in the presence of an intermediate communication channel or router failure.

(2) Flexibility to allow quick and easy topological changes.

(3) Ability to provide users with efficient service.

It is important to read between the lines of these requirements. Implicit is the assumption that routing is dynamic and decentralized, that both communications channels and routing hardware are somewhat unreliable, and that the number of routers changes often. This is in contrast to the telephone exchange-to-exchange network, where routing is quasi-static and centrally determined between large sites that almost never fail, and where the number of exchanges is relatively stable.

1. Routing has been approached as a graph theoretic question since work in this area began in the middle 1950's. Key works are Dijkstra's shortest path between two network nodes, determined from the network distance matrix [Dijkstra-59], and the distributed routing algorithm of Ford and Fulkerson [Ford-62].

The Requirement of Reliable Delivery

In a dynamic network, the route from one network site to another can change over time. In a decentralized network, this implies that routers must obtain information about network connectivity by exchanging routing update messages. This allows routers to change their choice of routes so that paths between source and destination are maintained when they might otherwise have been severed. The class of routing procedure chosen for a network determines both the number of updates exchanged, and the extent of their propagation throughout the network.

Nearly everyone agrees that a routing algorithm should make it possible to deliver data from one router to another whenever a path exists between the two. However, such a strong requirement is not always achievable. For example, it is not always practical in large networks to find a path between source and destination whenever it exists. The requirement is often weakened by allowing the routing algorithm to make the probability of delivery arbitrarily high in the general case but not to guarantee delivery in all cases.

Many computer networks have adopted a routing procedure called <u>shortest-path routing</u>. This procedure is popular because it provides maximum reliability in the presence of link and router failures; it can utilize all possible operating paths in a network. If a path exists between two routers, it can find that path. In this sense, shortest-path routing is the most reliable routing procedure. Shortest-path routing requires updated routing metrics (such as [illegible]tance or delay) to be periodically exchanged between all operating routers.

As a network grows, the volume of this update traffic grows, as does the time required to process, store, and generate it. This overhead can be overcome by slightly relaxing the requirement of reliable delivery and using a routing hierarchy. The resulting hierarchic routing still requires all routers to exchange update information, but the amount of information exchanged and stored is exponentially reduced. By using geographic forms of address, it is possible to further limit the exchange of update information.

The Requirement of Topological Flexibility

In a dynamic network, routers are periodically taken out of service, new routers are added, old routers are removed, and testing configurations are attached. The flexibility to make these topological changes is the second major requirement of any practical routing procedure. In a dynamic network, routers do not have fixed knowledge about all other routers in the network. Even fixed knowledge about immediate neighbors restricts flexibility, since it implies that neighboring routers must be halted and reloaded whenever a new router is added or removed. It also makes 'splicing out' a router for servicing impractical. Maximum topological flexibility implies that a router must be able to determine dynamically who and where its neighbors are.

The Requirement of Efficient Service

A network should provide adequate service to its users. Variations in user requirements, network topology, and communication bandwidth affect that service. The routes chosen should be adjusted dynamically, so that service to a user is rarely noticeably degraded. For example, as a region becomes congested, the network should avoid routing new traffic through that region if possible. This is usually achieved through the exchange of control messages among routers.

1.2 Characterizing Attack Resistance

A network is defined to be attack resistant if the results of any attack at a single point do not adversely affect network operation over a wide region for an extended period. Two types of attack are differentiated: *direct* and *indirect*. Direct attacks, such as a severed communication channel or a halted router, result in long-term damage to network facilities. These are the familiar forms of attack for which network operating software is normally designed.

Indirect attacks cause long-term damage by using characteristics of the network's own operating software to interfere with network operation. Examples are an invalid routing update that partitions the network, or a runaway router that saturates a region of the network. Network software is usually not designed with these forms of attack in mind. Both direct and indirect attacks can be the result of hardware failure, accident, or deliberate intervention. Ideally, network software should satisfy practical routing requirements and be attack resistant.

A network that has a single critical path, such as a local area network based upon a bus transport medium, is not attack resistant. Severing the coaxial cable between tap points at best partitions an IEEE 802.3 network and at worst halts communication, since the bus is then improperly terminated. For a network to be attack resistant, redundant communication paths are required. It must not be possible to partition the network either by halting any single router or by severing any single link between routers. The network must have a connectivity of at least two. To this is added the restriction that links use *physically separated* communications channels. A router whose links are separate channels within a single T1 group does not meet this requirement, since the links share the same physical transmission medium.

Consider a portion of a typical network, depicted in Figure 1. The types of direct attack possible at a single point in and around a router such as x are to disable x or to sever one of its links. Of these attacks the former is more severe and results in two types of damage: (1) x can no longer be used as an intermediary, and (2) x is no longer reachable. The first category causes short-term damage that can be corrected. By using a dynamic adaptive routing procedure, the immediate neighbors of x can initiate remedial action that causes the network to reroute around x. The second type of damage is long-term and cannot be corrected by the network. However, the long-term damage of the direct attack has been limited to the immediate vicinity of the attack.

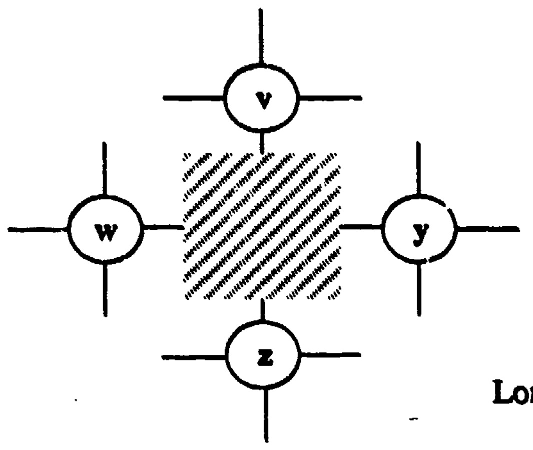
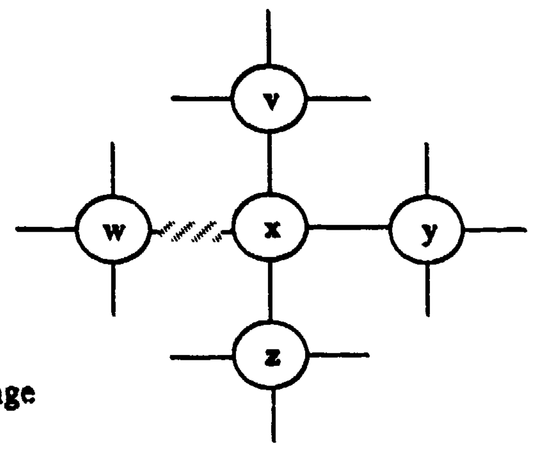


Figure 1.

Limited
Long-term Damage



A network can protect itself against direct attack by using redundancy in major network components and by providing multiple paths to each destination from any other router. Generally, networks take precautions against *direct* attack, but mechanisms to protect a network against *indirect* attack are less obvious. Indirect attack may be the result of hardware or software error, malicious assault on the network via insertion or modification of network messages, or alteration of a network router's software. The effects of an indirect attack vary, depending on network software. The mechanisms for protecting a network from indirect attack (if possible) are likewise not so straightforward as they are with a direct attack. However, one can formulate a guideline:

An attacker should gain no relative advantage via an indirect versus a direct attack.

This is useful as a design goal but it is unclear if it is achievable, since some indirect attacks may not even be detectable. Therefore, one requirement for an attack resistant network is that it be designed so that any single indirect attack is detectable if its long-term result more adversely affects network operation than any single direct attack. As an example, if a router generates a harmful and illegal routing update, its immediate neighbors should prevent the illegal update from reaching the rest of the network and possibly partitioning the network.

1.2.1 In Contrast to the Byzantine Generals Problem

An issue in distributed computing is the reliable transmission of information from a transmitter to a set of receivers in the presence of faults or malicious behavior. This is called the *Byzantine Generals* problem [Lamport-82][Dolev-82][Srikanth-87]. Assume that a network router z broadcasts a message to the other routers of the network. An algorithm that solves the Byzantine Generals problem satisfies the following two properties:

(1) All reliable routers agree on the same message.

(2) If z is reliable, then all the reliable routers agree on its message.

However, a Byzantine Generals solution does not by itself achieve attack resistance. If router x is *reliable*, it correctly executes the algorithm. The solution only guarantees that if x is reliable, then all correctly operating processors agree on the message x sent. No restrictions are placed on the contents of the message it transmits. To achieve attack resistance, it will usually be necessary both to restrict x's behavior and to ensure that reliable routers detect any violation of those restrictions.

1.3 Types of Indirect Attack

Messages are constructed by routers, whose hardware or software may fail or who may be subject to malicious outside intervention. Therefore, we must assume that any message, and in particular any network control message, may be subject to alteration or fabrication. Several questions immediately arise:

(1) What types of indirect attack can be made on a network?

(2) How does the choice of routing procedure affect the ways a network can be attacked and the extent of damage in the event of such an attack?

(3) What mechanisms can prevent such attacks or limit their damage?

We first discuss the types of indirect attack that can arise. We assume that router behavior or router-to-router communication departs in an unexpected manner from correct procedure. This may be due to hardware error, software error, or outside intervention.

1.3.1 Violation of Procedural Restrictions

Network designers usually assume that all routers cooperatively follow a mutually understood communications procedure. If this assumption is incorrect, trouble may ensue. Whenever any restriction is made concerning router or link behavior, the effects on network performance when the restriction is violated must be considered. It is not possible to list all restrictions, since this form of abuse is specific to each different routing procedure. However, we list two generally applicable restrictions, which when violated can severely damage most networks.

Misuse of Priority

Packets transmitted between routers for the purpose of network control are often sent with a higher priority than normal data packets. Routers usually process high-priority packets in their queues before processing normal packets. Otherwise, a flow-control packet sent to a congested router would only sit at the end of that router's packet-processing queue, adding to its congestion; or perhaps it might even be discarded.

High-priority control messages are exceptional by definition, and usually imply substantial amounts of computation on the part of a router. An example is a routing update that requires recomputation of routing tables. The combination of high priority and substantial router computation per packet provides an opportunity for abuse that could overload a router or greatly retard data-packet processing. Limitations on the frequency of generation of control packets are usually self-imposed by routers.

Invalid Control Messages

In a dynamic network, routers must exchange control messages in order to meet routing requirements. The exchange of control messages is what allows routers to adjust their routes to account for changes in delay and topology. Since routers themselves create most control messages, it is possible for routers to alter or fabricate them. An intelligent filter placed on a link could also do this. An invalid control message may cause routers to make incorrect routing decisions and possibly to supply incorrect information to other routers. Several possible damages could occur: non-optimal routes, increased delay, congestion, partitioning, and unnecessary routing computation. The amount of potential damage depends both upon the routing procedure's design and upon the nature of the invalid routing data. In some cases, the damage results in a severely partitioned network, where subsets of routers are unable to communicate with one another. If this is due to the absence of any communication paths between those subsets, it is unavoidable. However, incorrect routing information may cause partitions even though communication paths with adequate capacity do exist.

1.3.2 Masquerading as a Source or Destination

All traffic for network connections to separate hosts must pass through some router. All that connection traffic is subject to being monitored by a router. Furthermore, since each router can potentially access every host attached to the network, it can also synthesize traffic to or from those hosts. Much of this exposure of data can be prevented by employing end-to-end encryption, but this method may be impractical for many applications. Furthermore, in a commercial network, users might not bother to use end-to-end encryption, under the mistaken impression that the network is secure. For example, IEEE 802.3 networks use a broadcast medium, but there is little current commercial interest in designing network interfaces that allow the transparent enciphering of all station-to-station traffic.

Therefore, it is often possible for a router to monitor the end-to-end sequence numbers used by connection-oriented communications protocols, allowing the router to masquerade as either the source or the destination of a connection. Strictly speaking, except for the possible congestion that could be caused, this form of attack does not damage the network. Instead this attack damages the network's users. However, since the network exists to service user needs, this is an indirect attack on the network.

A router cannot mimic a source or a destination for arbitrary connections. Assume that a router is masquerading as the source of a connection. When a router mimics the source of a connection, return traffic from the destination will be routed toward the source's real address. When it mimics the destination, source traffic will be routed toward the destination's real address. This places constraints on any router that hopes to successfully masquerade as source or destination for an extended period.

In the simplest case, assume that the attacking router lies on the route the network has chosen to use for traffic to and from the affected hosts. As long as this situation holds, the attacking router can intercept all connection traffic for those hosts. This allows it to 'splice out' the connection's source or destination host whenever it chooses. It can then assume the role of the spliced-out host without an interruption in traffic. In principle, a destination host may never receive any packets for that connection, since the attacking router could mimic the connection initiation.

If the attacking router does not lie on the route taken by traffic between the affected hosts, then evidence of the attack will be available to either the connection's source or destination host. For example, if the attacking router mimics the source but does not lie on the route from the destination to the source's real address, the attacking router cannot intercept those packets; so the host at the real address will receive packets sent by the destination in response to packets it received from the attacking router. If the host at the source's real address is operating, it would see evidence of trouble on the connection, or evidence of a connection that never actually existed. A robust connection-oriented protocol would eventually respond to these situations by sending a connection-reset packet to the destination. If the destination received a reset, it would sever the connection, thus terminating the attack. A similar situation exists when the attacking router mimics the destination.

Routers must not be allowed to arbitrarily redirect traffic in a network. If a routing protocol allows this kind of redirection, it becomes much easier for a router to intercept traffic. From this perspective, a protocol in which each router makes specific inquiries concerning routes it may use, is preferable to a protocol in which each router broadcasts updates that influence a large number of other routers.

1.3.3 Discarding Packets

The most common kinds of indirect attack involve the creation of invalid control messages. These attacks are generally the most important to guard against, because their effects can be so far reaching. They can be detected because the exchange of invalid data provides direct evidence of an attack. However, an indirect attack that only deletes packets provides no such direct evidence and may be difficult if not impossible for a network to detect.

Assume that a single indirect attack is made by a router. Three classes of attack discard packets:

(1) Randomly discarding some packets received over a link.

(2) Discarding, in a non-random pattern, packets received over a link, for example, all packets from or to certain locations, or all those

except 'ping' packets sent to test transmission or delay characteristics of the link and attached routers.

(3) Discarding all packets received over a link.

Class 3 causes no special problem, since it emulates a severed link or a halted router, and normal network rerouting would limit long-term damage. In class 1, if only a small percentage of packets are discarded in a random pattern, a user would see a drop in effective bandwidth, but end-to-end retransmission would ensure that data was eventually delivered. However, classes one and two may not present any direct evidence of failure to the network in the vicinity of the attack. In that case, the network cannot be expected to take localized remedial action on its own.

If the percentage of discard is high, or if the pattern is not random (as in class 2), then retransmission will not correct the problem. The indirect evidence of such an attack would be severed end-to-end connections or the inability to initiate a connection. If an attack is malicious, then no remote network testing (i.e. testing that involves sending packets to/from the affected router) can be relied upon to provide evidence of the attack. The attacked router or link could merely mimic the correct response. Furthermore, the attack may interfere only with selected traffic that does not affect remote maintenance. Finally, the attack may be periodic or inconsistent, so that tests fail to detect the source of the attack.

If the pattern of attack is consistent, it may be possible to develop techniques for determining where the attack takes place. The point of attack would be located somewhere along the route taken by the affected end-to-end traffic. The controlled rerouting of affected traffic provides the only assured way of localizing the source of the problem. This type of routing adjustment is distinct from the kind of routing usually discussed. It must purposely avoid the primary route the network would choose, yet still prevent the creation of loops and congestion. Finally, if the point of attack becomes known, one facility must be available to the network administration: the ability to forcibly deconfigure a selected router or link from the network.

1.3.4 Generation of Artificial Traffic

Assume that destinations detect and discard unexpected packets. Any router can create artificial traffic for any destination toward which it can route packets. In so doing it could overload the network in its vicinity. Assume that the attacking router uniformly distributes the destinations. This could cause unnecessarily dropped packets and connections, and rerouting due to needless congestion. However, the network as a whole would not be severely damaged unless the router making the attack is on a critical path. Assume that sufficient excess network capacity exists to absorb the artificial traffic some distance from the point of attack. Since the destination addresses are uniformly distributed, the ratio of artificial to real traffic could be expected to fall rapidly with distance from the attack point. Without previously negotiated end-to-end flow-control restrictions, it is difficult for routers to detect or to control this type of attack at or near its source.

If the router creates artificial traffic with destinations chosen from a small set, then those destinations may become overloaded. This form of attack need not overload the network in the attacking router's vicinity, but may still overload selected destinations, since one generally expects hosts to have a much lower packet-processing capacity than routers. Thus, if certain destinations are critical to the performance of an important task, this form of attack may prevent that task from operating successfully. However, as before, this does not severely harm the network as a whole, and it is difficult to detect or to control this type of attack at or near its source.

1.3.5 Attacks Made on Links

The links between routers can be attacked in a number of ways that are analogous to attacks made by routers. They are:

- Inserting valid control messages.
- Inserting invalid control messages.
- Damaging or discarding packets.
- Masquerading as a source or destination.

Each of these attacks is considered separately.

Inserting Valid Messages --- Spoofing

One time-honored mechanism for attacking an information system involves the retransmission of messages. This 'spoofing' is accomplished by recording messages and then later injecting them into communications channels. Encryption will not by itself prevent this form of attack. By recording enciphered messages, the attacker can masquerade as the sender when he retransmits a message. It is not necessary for the attacker to know either the sender's key or his encryption algorithm.

This problem can be largely prevented through the use of sequence numbering or time stamps. Many communications protocols use sequence numbers to detect duplication, although this is usually done because senders retransmit after timers expire, not because spoofing is expected. Spoofing is a subject of current concern in the ARPA-Internet [Mills-87]. Undetected spoofing has varying effects; in extreme cases, it could cause complete network failure. Spoofing can occur in unexpected ways, as the following example shows.

Communications channels are modeled as point-to-point links connecting network routers. This is generally a gross simplification of the physical communication link and represents only its logical behavior from the point of view of the network. Networks often use 56–kb/s telephone circuits as their primary long–haul communication medium. The telephone companies, along the route the physical communications circuit actually travels, can make the circuit appear to a customer as if it were a dedicated, isolated, point–to–point link. However, between local exchanges in the U.S.A., twenty–four 56– or 64–kb/s circuits are usually multiplexed into a single T1 circuit. Twenty–eight T1 circuits may be multiplexed into a T3 circuit, and so on.

Two separate network links may therefore share a common circuit. The physical hardware that performs the multiplexing and demultiplexing sometimes has a transitory failure that causes frames from one circuit to be exchanged with another. From the network's point of view, this looks as if a packet is removed from the link upon which it was originally transmitted and is suddenly inserted into another link. Unless special care is taken, such an event may not be detectable.

These events, though rare, have occurred and have caused problems for networks. For example, the ARPA–Internet has two major long–haul components: MILNET and ARPANET. Valid routing packets between two routers of the ARPANET have appeared on links of the MILNET, presumably through this mechanism [Postel–87]. Since the MILNET and the ARPANET use essentially the same routing software, those packets seem valid to MILNET routers but contain very confusing and incorrect data.

Although spoofing can be prevented, its presence as a result of multi/demultiplexor failure is generally not expected. It is an excellent example of how an event entirely *outside* the network hardware and software designers' control could create a network failure. Situations such as this are obvious only in retrospect.

Inserting Invalid Control Messages

A filter placed on a link can selectively alter or insert messages, including network control messages. The effects on a network are similar to those that occur when a router creates invalid control messages.

Discarding Packets

A filter placed on a link can selectively alter or discard any packets that pass through it. The effects on a network are similar to those that occur when a router discards packets.

Masquerading as a Source or Destination

A filter placed on a link can potentially monitor all traffic that passes through it. Assume that the necessary information is not enciphered and that the desired connection traffic is passing through the filter. This would allow an intelligent filter to mimic the correct responses of the actual destination without discovery by the other end of the connection. For the same reasons it would be possible for the filter to mimic the source of the connection. The effects on a network are similar to those that occur when a router masquerades as a destination.

A filter could also insert artificial traffic. The effects of this are similar to those that occur when a router artificially creates traffic.

1.3.6 Remote Network Maintenance

Some networks are designed to allow a network administrative center to control and monitor routers. This is useful for examining patterns of network use, or for allowing the remote detection and correction of problems. One common capability is the ability to use the network itself (or a phone line) to load software, patch, and/or restart a router. The ability to remotely reload routers provides an avenue for attack. If a malicious user can reload router software, router function cannot be guaranteed. A solution to this problem is to never let software be reloaded. However, for router software to reside exclusively in read-only form implies a fixed software architecture. For a developing network with non-mature software, this is probably unrealistic.

Operating experience within the DARPA Internet suggests that remotely controlled loading may be best. Such capability must be carefully protected by authentication mechanisms, which are relatively easy to achieve, because there are few administrative centers compared to the number of routers. It is realistic to require passwords before granting internal access to each specific router, to assign keys, and to use end-to-end encryption of control messages sent by an administrative center. This nearly eliminates the possibility that invalid software or commands can be sent by some source masquerading as an administrative center. Storage of the administrative password and key tables can be restricted to one or a few centers where their use is carefully monitored. Message replay should be prevented by using time-stamps and sequence numbers.

1.3.7 Misuse of Multicasting

Multicasting allows a subset of destinations to be identified as a group. It has become increasingly popular with the spread of local area networks, since many of them support multicasting or broadcasting within their cable segments. A multicast group may also be considered as a subset of destinations with no relationship to any particular subnetwork. Membership in the multicast subset may be dynamic, with specific destinations allowed to join or leave the subset. Membership may also be regionally defined.

A single multicast packet potentially causes many packets to be delivered. Special actions are taken to implement the transmission and reception of multicast packets. In general, a multicast packet requires more computing within routers than does a standard data packet. It also requires each recipient in the destination subset to examine a copy of the packet.

Transmission of a multicast packet by a host is infrequent relative to standard data packets. Their use is limited to those occasions where their one-to-many characteristic provide a distinct advantage. This recognizes their expense both to the network and to the members of the multicast group. If multicasting is abused, portions of an internetwork could become overloaded as could the members of multicast groups. The limitation usually placed upon the transmission of multicast packets is 'good network citizenship.' Both routers and hosts can abuse this facility.

1.3.8 Server Malfunction

As network software becomes more sophisticated, the number of remote services grows. Some of these services are now necessary for correct network operation. When servers are important to basic network operation, they provide an avenue for indirect attack. An example of such a service is the Name-to-Address mapping now being used in some large networks. The Internet version of this is the Domain Name Service [Mockapetris-83].

Users of the service often know of only one server, their initial contact, and rely on that server to provide needed information. The server usually responds to user requests either by referring the user to other more appropriate servers or by returning the requested information itself. The location of these other servers is sometimes determined by a discovery mechanism.

If some server possessing only limited information masquerades as a source of more global information, requests for information may be 'short-circuited.' The correct server may never receive the enquiry, a server may claim it has the requested information when it does not, or false information may be returned to the user. That portion of the network serviced by the malfunctioning server is denied access to correct information. In the case of a domain name server, this can effectively partition a network. This has already occurred at least once in the Internet, when a server incorrectly claimed to be a source for information that it did not possess [Lenoil-87]. The result was an inability to translate names to addresses for a portion of the network.

2 Examples of Protocol Behavior in the Presence of Invalid Routing Data

One category of attack affects all dynamic routing procedures: the circulation of invalid routing data. Assume that one network router receives a single control message containing invalid routing information. We will discuss the extent of potential damage in the context of three commonly used routing procedures that meet practical routing requirements:

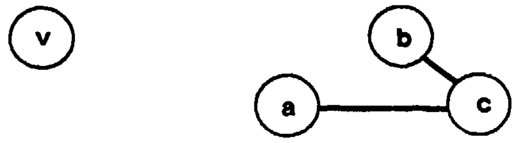
1. Internet routing.
2. ARPANET 'shortest-path-first' (SPF) routing.
3. Hierarchic routing.

Of these three, the Internet and ARPANET procedures are both true shortest-path procedures. They represent a class of procedure that is heavily used today, but neither is suitable for large-scale applications. Hierarchic routing uses a modified form of shortest-path procedure that is suitable for large-scale application.

2.1 Internet Routing

The Internet uses a shortest-path routing procedure that measures distance by counting the number of gateway-to-gateway hops; the distance to any immediate neighboring gateway is one unit [RFC 823]. Thus, for a particular source-to-destination path, a cost of ten implies that ten internetwork gateway-to-gateway transitions occur on the path to the destination. Although simple, this procedure has several drawbacks, and it has been suggested that it be replaced by the ARPANET SPF procedure. Those interested in this topic should read Rosen's comparison of the two algorithms [Rosen-81-1].

Figure 2.



2.1.1 Invalid Routing Updates

We ignore the effect of the intervening networks between gateways (as does the Internet routing procedure) and model the Internet as a set of routers connected by links that represent the intervening networks. Figure 2 shows a fragment of this network in which routers **a** and **c** are immediate neighbors. Events such as the addition of a new router or a new link, the return of a link to service, or the removal of a link, cause a router to create updates that are sent to its neighbors. An update sent by router **a** to a neighboring router **c** contains a table of the current shortest distances from **a** to all other destinations in the network for which **a** believes it is as close to or closer than **c**.

If router **c** receives an update from its neighbor **a**, it examines the update to determine whether or not it is different from the last one received from **a**. If the update is not different, it is discarded and no action is taken. If it is different, then **c** takes the following actions:

1. It recomputes its shortest-path routing table.
2. For each neighbor, including **a**, it computes and sends an update.

Updates are exchanged until no router receives from any neighbor any new routing information. At that point each router knows the next hop taken on the shortest path to any non-partitioned network destination.

Network Partitioning

Let the function $C(x,y)$ represent the delay or cost of communication between x and y as seen from x's point of view. The actual values represented by $C(x,y)$ fluctuate throughout a network as topology and communications patterns change. Let $R(x,y,C(x,y),z)$ represent the sending of a routing update from x to z; this update informs z of the cost to communicate from x to y. Generally, the values $C(x,y)$ received by a router cannot be verified without additional knowledge. In the absence of that knowledge, a router must trust such values.

Assume that $C(\mathbf{c},\mathbf{v}) \gg 1$. Consider the effect if router **a** sends $R(\mathbf{a},\mathbf{v},1,\mathbf{c})$. Router **c** will now calculate a temporary cost of $C(\mathbf{c},\mathbf{v}) = C(\mathbf{c},\mathbf{a})+1$. This value is near the lower bound on possible values for $C(\mathbf{c},\mathbf{v})$ if $C(\mathbf{c},\mathbf{a})$ is near 1. To **c** this will normally indicate that the path to **v** that starts with the first hop $\mathbf{c} \rightarrow \mathbf{a}$ is its best path to **v**. This incorrect assumption causes **c** to change its routing table. The new table will direct traffic that is eventually intended for **v**, and which passes through **c**, to **a**. Furthermore, **c** will send its updated routing data to other routers as required by the routing procedure. This process will quickly propagate the incorrect routing information.

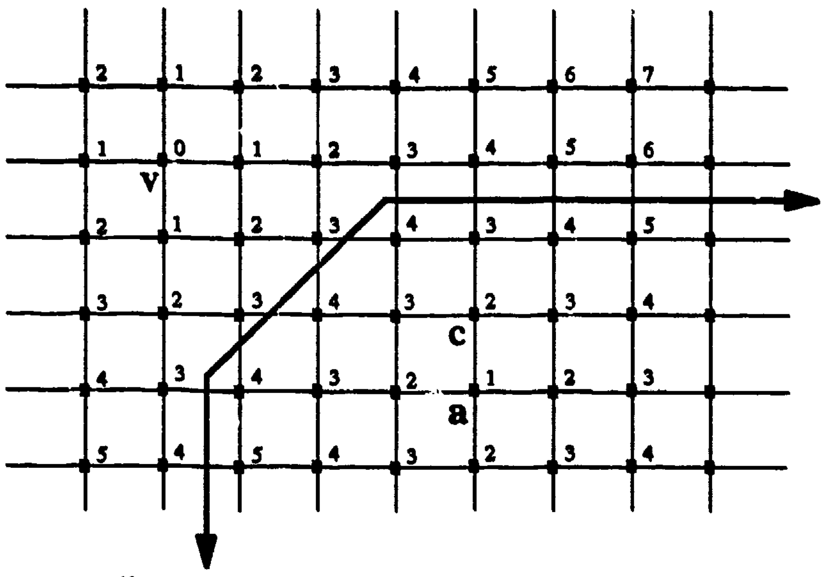
The extent of update propagation depends upon the values in the update, current costs on links, and current network topology. For demonstration, assume that the network topology is a grid, as in Figure 3. Router **a** now alters its routing table so that $C(\mathbf{a},\mathbf{v}) = 1$ and sends $R(\mathbf{a},\mathbf{v},1,\mathbf{c})$, where **c**'s previous cost to reach **v** was six. The value $C_{tmp}(\mathbf{c},\mathbf{v}) = C(\mathbf{c},\mathbf{a})+1 = 2$. This is smaller than six so **c** will alter its routing table and send $R(\mathbf{c},\mathbf{v},2,k)$ to all of its immediate neighbors k.

This process continues until routers receive updates that indicate they are as close to **v** via their current path as they would be by the path via **a**. Those routers do not need to change their routing tables, so they cease to propagate the incorrect information. In Figure 3, the resulting costs to reach **v** are placed beside each router. Clearly marked is the boundary beyond which routers do not change their tables. The network has been partitioned into one section which can reach **v** and another which cannot. All routers within the partition containing **a** will forward data for **v** toward **a**, resulting in useless congestion.

The incorrect value sent by a router need not be as small as one. Any incorrect $C(x,y)$ value that is smaller than the values of $C(x,y)$ for all its immediate neighbors

Figure 3.

Partitioning



causes partitioning. Sending a smaller value creates a larger partition. Once established, an incorrect path-set is stable. Only by disconnecting the router that sent the invalid update, implying C(**a**,v)=∞, or when **a** resends the correct value for C(**a**,v) = 7, can the network be restored to its correct operation. In the Internet procedure, a gateway can send many incorrect C(*x*,*y*) values in a single packet. The transmission of such a 'black-hole' packet would rapidly cause the network to cease effective operation.

Persistence of Damage

A partition caused by incorrect routing information will remain until contradicted. The router **a**, which has become the center of this type of partition, will detect the presence of the partition when it receives an erroneous update (assuming that it was not itself the cause of the error). Once **a** detects the problem it must transmit the corrected R(**a**,*i*,C(**a**,v),*k*) to all neighbors *k*, and to all routers *i*, for which invalid topological data was supplied. This ensures the eventual correction of the partition, but only if the attack was accidental.

A router that is being used to make a malicious attack on a network cannot be expected to act in the network's interest. Under this circumstance, a mechanism that removed the offending router from the network would be the only one that could reliably restore correct network function. Unfortunately, **a**'s neighbors cannot determine that a partition is caused by an invalid update sent by **a** without additional information. Without better lower bounds for C(**a**,v), most values supplied by **a** seem reasonable. Furthermore, detecting the source of a bad routing update may be difficult. If c were malicious it could create a routing update that makes it appear as if c had just received an update from **a**, claiming that **a** had a link to v.

2.1.2 Detection Does Not Imply Prevention

In certain cases routers may be able to determine that a particular control message is invalid. In Figure 2, when v receives a routing update that erroneously claims **a** is connected to v, then v can determine that the update is invalid. It seems that v merely needs to broadcast to all routers the fact that a's update is invalid to prevent any damage. However, in the general case there is very little v can do to remedy the situation.

First, v must be able to determine who actually generated a particular message, which may not be possible. Second, allowing a router to send a message claiming that an update from another router is invalid would make the network more vulnerable to malicious attacks. If this were allowed, a malicious router would not need to send invalid updates; instead, it could claim that other routers had sent invalid updates. This technique could be used to freeze the routing state of other routers, eventually producing static routing. Finally, by the time v received the invalid update, much of the partitioning may have already occurred. The resulting routing and congestion problems may make it difficult for corrective messages from v or an administrative center to reach the affected routers.

2.2 ARPANET Routing

The shortest-path first (SPF) routing procedure used in the ARPANET contrasts strongly with that used in the Internet [McQuillan-80][Rosen-80]. Each router periodically measures the round-trip delay to its immediate neighbors and uses that as a measure of the cost to those neighbors. In the ARPANET, when a router builds an update, that update contains only the identity of the router's neighbors and its measured delay to them. The update is then broadcast by flooding to all other routers in the network. This information is employed by each router to construct a shortest-path spanning tree of the network, with itself at the root. Convergence to a network-wide consistent solution is supposedly more rapid in the ARPANET than it is in the Internet, and the amount of information exchanged is much smaller.

2.2.1 Invalid Routing Updates

At first glance it appears that the ARPANET routing procedure is protected from the distribution of invalid routing data. A router x can lie about delay only to its immediate neighbors. This may cause other routers to create non-optimal spanning trees that use x, but it is unlikely to cause severe partitioning. However, the protocol allows a router to report a connection to a new neighbor. Furthermore, updates are distributed via flooding, and a router can generate fictitious updates that appear to have been originated by another router. Returning to Figure 2, this implies a can falsely claim that it possesses a low-delay link to v, or a can generate a false update claiming that some other router has a low delay link to v. Since ARPANET routing is another shortest-path procedure, the spanning trees calculated by the other routers in the network will produce partitioning similar to that which took place in the Internet.

The damage is limited by the choice of delay rather than hop-count as a measure of cost. If congestion due to partitioning causes sufficient increases in delay, then within the partition the implicit flow control associated with this routing update procedure will redirect paths away from the congested area.

2.3 Hierarchic Routing

Hierarchic routing, first analyzed by Kamoun [Kamoun-76] [Kleinrock-77], divides addresses into fields. Each field is associated with a particular level in the hierarchy. Each router resides within one particular lowest level cluster of routers in the hierarchy. Adjoining lowest level clusters are grouped into higher level clusters; these in turn may be grouped into yet higher level clusters, and so on. A router knows specific routing information about each router in its own lowest level cluster, and less specific information about each successively higher level cluster.

Figure 4 shows the routing table for a three-level hierarchic network. Router [1.1.1] resides in a particular level-0 cluster with other routers [1.1.2], [1.1.3], and [1.1.4]. It is part of the higher level-1 cluster [1.1.], which is in turn part of the higher level-2 cluster [1.]. The entries marked with asterisks are unnecessary as they represent router [1.1.1]'s distance to its own clusters.

The routing table is similar to the shortest-path tables discussed in sections 2.1 and 2.2. Router [1.1.1] knows specific information about paths to other routers within its own lowest level cluster. Within successively higher levels, less is known about *individual* routers within a cluster; a router knows only a shortest path toward *some* router inside clusters above level-0. For example, router [1.1.1] knows the next hop on the path to clusters [1.2.] or [3.], and the distance associated with those paths. It does not know paths to specific routers in those clusters.

As packets approach their destinations, more precise routing information does become available. For example, a packet from a router within cluster [2.], destined for [1.1.1], will eventually reach *some* router in cluster [1.]. That router knows a path toward cluster [1.1.]. When the packet reaches a router in cluster [1.1], that router knows the final path to [1.1.1].

Hierarchic routing solves problems of scale associated with shortest-path routing: routing table size, routing decision time, and update processing time that grows with the number of hosts. Hierarchic routing retains the capability of finding short paths under

	Destination	Next Node	Delay	Hop Count	
	1.1.1				*
Cluster level 0	1.1.2				
	1.1.3				
	1.1.4				
Cluster level 1	1.1.				*
	1.2.				
	1.3.				
Cluster level 2	1.				*
	2.				
	3.				

Figure 4.

Routing table for 1.1.1

most circumstances by requiring routers to exchange updates. Update information does propagate throughout the network. Therefore, a router can send invalid updates that affect the entire network. In particular, a router can claim it has a short path to a high-level cluster. For example, [1.1.1] could erroneously claim it had a path of length 1 to some router in cluster [2.].

2.3.1 Dynamic Organization

Several large-scale hierarchic organizations have been proposed for computer networks. Details vary with each proposal, but in general, for the purpose of managing the exchange of routing information within the hierarchy, controlling cluster membership, or directing the routing itself, a group of routers elects a representative that is principally responsible for its group. These representative routers have been variously called coordinators, leaders, or cluster heads. In a real network coordinators will eventually fail due to hardware failure or software errors. This raises the question of their dynamic assignment.

One solution to the problem of dynamically choosing a coordinating router is to use a parallel election algorithm [Garcia-Molina-82]. Election or invitation algorithms assume that a number of network properties hold for their correct operation:

- All routers cooperate, and there are no software errors.
- If router *i* receives a message that claims to come from router *j*, then router *j* did indeed send that message.
- When a router fails, it immediately halts processing.
- A router does not behave in an unpredictable manner.
- There is no tampering with data in any messages sent between routers.

A temporary hardware failure or malicious attack could violate each assumption. There is no guarantee either that a particular group of routers will elect a single coordinator, that any group coordinator will eventually be chosen, or that a router will know within which group it resides. For example, one router could ignore the election of another router as coordinator, and arbitrarily assume the coordinating role itself. To be attack resistant, a hierarchic network that uses election algorithms must demonstrate that any damage resulting from the failure of the election algorithm would be limited to the immediate vicinity of the offending router.

2.4 ANSI/OSI Hierarchic Routing

A variant of hierarchic routing is the Intermediate System-to-Intermediate System (IS-IS) routing procedure within the OSI reference model [ANSI X3S3.3]. Under this model a three-tiered system is proposed. Tiers in this proposal are each identified with one or more hierarchic levels, corresponding to regions of differing administrative control. Tier one is identified with hierarchic level 0, and performs End System-to-Intermediate System (ES-IS) routing. Tier two is identified with levels 1, 2, ..., 2+*i*, in which one or more *Clusters* comprise a *Domain*. Tier three is identified with levels 3+*i*, ..., 3+*i*+*j*, comprising a *Dominion*. Above this (although still considered to be in tier three) is the *Common Dominion*. A minimum of four hierarchy levels is allowed. Hierarchic routing is proposed for tiers two and three. Tier one is considered small enough that flat routing may suffice.

A Domain uses a common set of routing procedures. It may or may not be *transitive*, in the sense that it will act as an intermediary between other domains. A Dominion is differentiated from a Domain in that it implements a set of routing policies prescribed by its routing authority. These policies control the ingress and egress of information crossing the Dominion's boundary. At the Common Dominion-level, all routing policies must be arrived at via a multi-lateral agreement reached between all participating Dominions.

2.4.1 Non-Uniformity of ANSI/OSI Routing

The proposed ANSI standard suggests that Common Dominion routing be determined at the Application Layer rather than the Network Layer. Before communication begins between applications, extensive end-to-end negotiation would take place to exchange authentication and billing information. This sets it strongly apart from routing procedures we have discussed earlier. The routing database maintained for Common Dominion-level communication is considered sufficiently stable and infrequently needed that it can be obtained from a System Management authority. The routing algorithm would be Static/Quasi-static as opposed to Distributed Adaptive.

Routing at the Dominion level and lower should allow both Static/Quasi-static and Distributed Adaptive routing as options. The Distributed Adaptive routing procedures discussed, although not specified, strongly resemble the ARPANET and Internet routing procedures in intent and mechanism. Therefore, it is at the Dominion level and below that dynamic routing procedures may provide opportunities to damage the network.

3 A Detailed List of Possible Attacks in Two Routing Procedures

The circulation of invalid routing data is only one common way a distributed dynamic network can be attacked. Many other forms of attack exist, but these are usually specific to particular network procedures. To illustrate this, we list some of the ways two specific procedures can be attacked. Assume that a router violates procedural restrictions, and that it acts alone. Cooperative effects that arise from simultaneous violations at more than one point in the network are not discussed.

3.1 ARPANET (SPF) Procedure

Some kinds of attack can damage a network much more severely than the damage that results from the circulation of invalid routing updates. The ARPANET SPF procedure provides an example. The restrictions for the ARPANET procedure were derived from reports on ARPANET routing algorithm development [Rosen-80][McQuillan-78-2]. The following discussion assumes familiarity with the ARPANET routing procedure. The relevant portions of those references are summarized in Appendix I for the reader's convenience. Some of the numeric values discussed may have changed since the references were published.

3.1.1 Effects when Violating ARPANET Routing Procedure Restrictions

In this section we will briefly discuss the effects on the network when an IMP violates the routing procedure restrictions in the ARPANET. We assume that the misbehaving IMP acts alone. Cooperative effects that arise from simultaneous violations among more than one IMP are not discussed.

Computing the Wrong Spanning Tree or Choosing the Wrong Outgoing Link

After receiving an acceptable update, an IMP calculates a new lowest cost spanning tree for the network. This is used to choose the outgoing link for each destination. A central assumption of the routing procedure is that all non-partitioned IMPs calculate their spanning trees from the same data. Failure to meet this restriction results in network-wide inconsistencies. The same result occurs if the tree is calculated incorrectly. If an isolated IMP calculates its tree incorrectly, then it will choose the wrong outgoing link for a *set* of destinations.

Assume that IMP *i* chooses the wrong outgoing link for a destination. Either IMP *i* sends to neighboring IMP *j*, and *j*'s decision produces a route back to *i* resulting in a tight loop, or *j*'s decision results in another route that avoids *i*. In the first case, a destination may appear unreachable, and a partition has been created. Both routers could become overloaded while the loop persists. This situation can be detected and avoided by *j*. In the latter case, the detrimental effect is a loss of efficiency.

Age-Value Field

The age-value field limits the time any particular update can reside in the network before being discarded. Properly chosen, this ensures that, when a partition occurs and after the age-value limit expires, no pre-partition updates from one part of the partition are circulating in the other part of the partition. Also, two updates from the same IMP must not be allowed be circulate in the network simultaneously for long enough that their serial numbers wrap around. With a 6-bit serial number field, this occurs every 32 updates.

At the maximum update generation rate of 12 per minute, 32 updates require 160 seconds. The age-value field, in conjunction with its associated clock, must be used to discard any updates substantially before they are 160 seconds old. The maximum age-value that an update packet can specify is eight, and an IMP age-value clock ticks every eight seconds. Thus the maximum age an update packet can reside in the network is 64 seconds plus network transit time. (Transit time is nominally -100 ms and is therefore negligible for this purpose.)

An IMP can violate age restrictions in essentially two ways:

(1) Not use the maximum age-value for updates it originates.

(2) Alter the age-value field of updates it transmits.

If the IMP does not insert the maximum legal age-value into the updates it originates and instead inserts a smaller value, then possibly some of its updates will have aged sufficiently that they may not be transmitted by IMPs to their neighbors across a partition when that partition ends. If this occurs it can result in a temporarily inconsistent network-wide routing database.

An IMP can alter the age-value of updates it transmits via flooding. This can occur selectively, or the entire age-value vector could be altered in three ways: up, down, or in such a way that the values are not decremented when the clock ticks occur. If age-values are decremented, effects will be similar to age violation (1) above.

Assume that IMP *j* keeps an old update from IMP *i* in its database after the update's age-value would normally have decremented to zero. Assume that *j* retransmits the update from *i*, while *i* transmits a new update with a different serial number. This results in two updates that originated from one source, circulating in the network simultaneously. If sufficient time has elapsed during the partition, serial numbers may have wrapped around, and the old update from *j* may appear more recent than the new update from *i*. The old update could circulate throughout the network, overriding the new update and impairing performance. This performance degradation could persist for many minutes.

Serial Number

Serial numbers within updates are used by IMPs to determine the relative age of an update. If a newly received update is more recent and its age-value is non-zero, it replaces the older update in the IMP's internal database. By altering serial number values in the updates it transmits, an IMP renders useless the test for relative age.

If IMP *i* alters upward a serial number for one of another IMP *j*'s updates from value k to value l and retransmits that update, then a series of updates from IMP *j* will be ignored by a portion of the network until IMP *j* issues an update with a serial number more recent than l. Other IMPs will ignore those updates in the range [k+1, l]. IMP *j* can notice this alteration if it receives a copy of the update with serial number l. IMP *j* at this point knows that something is wrong somewhere in the network with some IMP (possibly itself). Its safest strategy would be to wait until the age-values of the update it most recently sent decrement to zero, and then to retransmit using serial number (l + 1).

If IMP *i* alters two or more serial numbers from another IMP *j* and retransmits them, then the possibility exists for an infinite cycle of updates. Assume that three updates originating from IMP *j* exist simultaneously in the network. Whenever an IMP creates a new update it increments its serial number. Serial numbers are limited to the range 1-to-63, with zero specially reserved. When incremented past 63, they begin again at 1. When an IMP receives two updates from the same origin with two serial numbers $x \neq y$, it must determine which is the more recent. If either $y > x$ and $y-x \leq 32$, or $y < x$ and $x-y > 32$, then y is assumed to be more recent than x. Otherwise x is assumed to be more recent than y. When an IMP accepts a more recent update, that IMP immediately retransmits the update to its neighbors.

Using that determination, there exist sequences $x < y < z$ that form a cycle, with y more recent than x, z more recent than y, and x more recent than z. An example is {20, 30, 60}. The simultaneous existence of updates originating at the same IMP, with serial numbers that satisfy these restrictions, can create an indefinite cycle of update acceptance and retransmission. Since updates are processed by IMPs at higher priority than normal data messages, this can effectively halt the network.

On October 27, 1980, just such an event occurred in the ARPANET. It resulted in a catastrophe that disabled the entire network for several hours [Rosen-81]. There were four contributing causes:

1. Hardware failure. A section of an IMP's memory became unreliable.

2. Human intervention. The parity-checking circuitry was disabled, and an IMP was allowed to continue operation because human operators observed that most parity faults were spurious.

3. Physical IMP implementation was slow enough to encourage short-cuts. Internal routing table data was not protected by checksums. As a result, corruption of internal data was not detected and that data was distributed.

4. The routing procedure. Routing data is exchanged among all active IMPs in a way that allows a cycle to develop. This resulted in the continual redistribution of routing information. The resulting CPU demands upon the IMPs caused them to cease effective operation.

Those factors combined to cause one particular IMP to transmit updates with the sequence numbers 8, 40, and 44, in that order. This is a cycle, and it caused the three associated updates to flow indefinitely around the network at high priority. As a result, all the network IMPs expended most of their CPU time processing and retransmitting routing update messages.

It required some time to diagnose the problem and nearly four more hours to correct the problem and restore the network to normal. The source of the problem was narrowed to one of two IMPs that had experienced intermittent hardware errors. It is important to notice that all other IMPs that received these updates were operating correctly. Halting the IMP thought to be the source of the problem was not sufficient. The network control center (NCC) had to load into each IMP a code patch that directed it to ignore updates from the damaged IMP. Under normal conditions this would take only a few minutes, but under these circumstances it took hours. It was fortunate that the ARPANET had only approximately 50 IMPs and not several thousand.

It is clear that many such update sequences can be found. This occurred entirely by accident, from an unlikely set of circumstances. Network designers did not consider it a serious possibility. However, a malicious router could easily create this situation and halt the network. Such an attack would be extremely damaging, difficult to prevent, and difficult to correct once it occured.

Update Frequency

An IMP must generate an update at least once per minute, and no more than 12 times per minute. If IMP *i* generates updates at too low a rate, its updates will have their age-values decremented to zero. If a partition ends, this could result in valid updates from *i* not being transmitted across the former partition boundary. This in turn may lead to network-wide database inconsistency.

If updates are generated too frequently, the overhead associated with processing them throughout the network becomes excessive. Updates are processed at a higher priority than normal data messages, and each acceptable update also implies substantial computation on the part of the IMP that receives it. Furthermore, they are distributed by flooding to all network IMPs. A sufficiently high generation frequency of acceptable updates would effectively halt the network.

Timers

Timers are used by IMPs to trigger age-value decrementation and frequency of update. Timers are usually driven by a hardware clock source. A malfunctioning clock can produce violations involving age-value or update frequency. If the clock runs too slowly, then age-values will not be decremented when they should, relative to other IMPs that have properly functioning clocks. Similarly, updates may not be generated as often as required. If the clock runs too quickly, then age-values will reach zero too soon and updates may be generated too often. The possible effects of these events were discussed above.

If the restrictions involving the timers in IMP *i* associated with its input lines are not followed, then either updates will be unnecessarily retransmitted to neighbors or not retransmitted when necessary. In the former case, excessive delay and overhead may occur in the immediate vicinity of IMP *i*. In the latter case, one of *i*'s neighbors, *j*, might not receive a good copy of an update. Unless *i*'s removal would partition *j* from the network, flooding retransmission by other IMPs would ensure that *j* received a copy.

Refraining from Flooding

When IMP *i* judges an update to be acceptable, that update's relevant data must be stored internally, and then the update must be retransmitted to all *i*'s immediate neighbors. One way *i* can violate routing procedure is to selectively block retransmission. In a well-connected network this would not cause great harm.

3.1.2 In Conclusion

The ARPANET routing procedure requires all IMPs to simultaneously obey a number of restrictions. Failure to meet these restrictions can result in network-wide failure. Irrespective of other forms of attack (such as invalid delay data), the ARPANET procedure provides little or no protection from malicious attack. Furthermore, as we discovered in the October 1980 incident, deliberately malicious behavior is not required. Network-wide failures can result from relatively simple, short-lived, transient errors in hardware.

3.2 Cartesian Routing Procedure

The Cartesian routing procedure is a relatively attack-resistant routing procedure [Finn-87]. We will examine it in contrast to the ARPANET procedure. Under Cartesian routing, addresses are two-tuples that separate location and identity. From the location portion, a distance can be calculated from any router to any address, whether fixed or mobile. The Cartesian procedure is not based upon shortest-path techniques, as were the others previously discussed. It does not periodically propagate routing information throughout the network. Instead, routers determine from a packet's destination address whether or not a neighboring router makes progress toward (is closer to) that destination. If a neighbor is found that makes progress, then the packet may be forwarded to that neighbor. If no progress is made, a router initiates an enquiry in the surrounding region of the network for any router that makes progress. This is accomplished by means of flooding, where the radius of the enquiry is limited by hop-count. This flooding-limit **f** is usually a small integer.

The key difference between this procedure and others modeled on shortest-path techniques is that routing information is not broadcast throughout the network when a topology change is noticed. Instead, routing information is specifically requested by a router when it cannot make further progress toward a packet's destination. In large Cartesian systems, the network may be divided into a hierarchy in which each higher level possesses communication links of greater hop-by-hop distance. This may be used to keep hop-counts low when the distance between source and destination is large, and to route around large topological irregularities.

3.2.1 Maximum Hop-Count Flooding Limitation

When a router cannot make further progress toward a specific destination, it initiates a flooding procedure, which searches for a source route to another router that is closer to the destination than itself. The spread of flooding requests is controlled by limiting the range they propagate from the initiator to no further than a specified number of hops, f. Routers that receive a flooding request packet add their address to a source route list in the packet header. If they are closer to the destination, they respond positively to the requestor with the source route between themselves and the requestor. Otherwise, if they cannot reply successfully and are less than f hops distant, they forward the request one hop farther. From zero to many responses may result, with each response containing a source route of length $\leq$ f+1. If no response is received, then after a predefined period the destination is assumed to be unreachable. A detailed description may be found in [Finn-87].

For a network of N routers, where on average each router has C immediate neighbors, the upper bound on the number of routers that could possibly receive a routing request from any particular router is limited to $C(C-1)^{f-1}$. A router supplies routing information on request. Its response can only be seen by those routers along the path back to the requestor, which is at most f routers. The number of routers potentially affected by an invalid routing update conveyed in a single response is therefore strictly limited. One measure of network susceptibility to the supply of invalid routing data is the ratio of the maximum number of routers affected by an update to N. For a Cartesian procedure and a large network $f/N \ll 1$. For hierarchic and shortest-path procedures the ratio approaches one.

Drawbacks of a Maximum Flooding Limit

In some situations, the qualitative cost of a fixed flooding limit may outweigh the protection advantage gained. Consider the situation in which a region of the network has been extensively damaged, requiring a flooding limit greater than f in order to open connections that proceed around the damaged region. A more robust variant of Cartesian routing would have no absolute enforced network flooding limit, instead allowing a user to exceed the normal flooding limit for certain connections or traffic categories. A slightly more restrictive option would allow the flooding limit of local routers to be adjusted by a local routing authority. These options would be of value to a military network.

3.2.2 Effects of Violating Cartesian Routing Procedure Restrictions

Some of the effects that result when a router violates Cartesian routing procedure restrictions are discussed in this section.

Violating the Flooding Procedure

A single router may violate the limited flooding procedure in several ways, and those violations may have a number of effects on the network. A router may:

- route a packet on a path that makes no progress.
- ignore the flooding limit.
- ignore or alter the flooding response time limit.
- not place its address into the source route list.
- illegally alter the source route list.
- fail to reply successfully when it should, or reply when it should not.
- fail to forward a flooding enquiry when it should, or forward one when it should not.

Forwarding to Routers that Make No Progress

The most basic routing error is to choose the wrong entry from the routing table or to ignore the table. Assume that a router *h* chooses to forward a packet to an immediate neighboring router *j*, whose location is not closer to the destination's location. This could result in a loop, with the packet being returned to *h* (possibly by *j*).

Router *h* may also choose an incorrect source route from its tables, or create one and use it to forward the packet. This is equivalent to an intermediary router that alters the source route of a packet passing through it. If the resulting source route is illegal (no such route exists), that problem will be determined en route, and the packet will be discarded. Otherwise, the packet will reach the end of the source route. The routing decision made there may result in a loop.

Sometimes this alteration can be detected and avoided. Any intermediary or terminating router can examine the progress limit field, the last address on the source route, or the destination location, and determine whether or not the packet makes progress toward its destination. In a single-level Cartesian network, any packet that fails this test may be immediately discarded, since the rule of progress has been violated.

Assume that router *h* is the first router on a source route, *i* is an intermediary router, and *j* the last. In a multi-level Cartesian network *h* and *j* are on either the same or adjacent levels of the network routing hierarchy. If router *j* resides on the same level of the network as router *h* or below it, then any packet that fails this test may be immediately discarded by *i* or *j*, since the rule of progress has been violated. If *j* resides in a level above that of *h*, then the packet should not be discarded by *i*. If a packet fails the test and is not discarded, it must have moved up a level to be forwarded there. The maximum number of this type of routing error per packet is fixed, since each Cartesian network has a fixed number of levels.

Ignoring the Flooding Limit

A flooding limit field is included in the packet header that is set by the flooding initiator. This field's value is decremented whenever a flooding request packet is forwarded one hop further. A router could illegally modify that value upward or downward. When the value is decreased, the extent of the flooding search is diminished. This could result in needlessly dropped connections. When it is increased, the limit of the search is extended beyond the initiator's requested limit. This adds to network overhead but in some cases may actually prevent dropped connections.

The flooding limit can be enforced in a number of ways. An overriding network upper limit F should be incorporated into the code of each router. If all routers examine the field value of flooding packets that pass through them to ensure that the packet limit field value of $f \leq F$, then the most a single router can change the value is from one to F. The length of the source route list could be limited in a similar manner.

Altering the Progress Limit Field

The progress limit field is used by a router, which has received a flooding request packet, to determine if it has a neighbor that is closer to the destination's location. If there is such a neighbor, the flooding procedure is successfully terminated. If a router decreases the limit field, flooding may be unsuccessful when it would otherwise have terminated successfully. The possible results of this are:

- sending notification of destination unreachable to the source
- choosing a less desirable route
- rerouting packets up-level needlessly in a multilevel network

If the limit field is increased, flooding may be successful when it would otherwise have failed. This could result in a loop but can often be detected and avoided through examination by intermediaries, as shown above in the section on forwarding to routers that make no progress.

Limiting the Flooding Response Time

The Cartesian routing procedure relies on a clock for only one thing: determining when to stop waiting for flooding responses. If this time limit is ignored or the clock is too slow, end-to-end protocols would time-out, retransmit, and eventually assume that affected destinations are unreachable. This would add some small amount of network overhead. If the time limit is reached too quickly or the clock is too fast, possibly usable flooding responses may be ignored. The effects of this are similar to those of decreasing the progress limit field, discussed above.

Omitting an Address from the Source Route List

When a router receives a flooding request packet, it is required to append its address onto a *flooding router list*. If a successful flooding reply is generated, this is a reverse source route to the flooding initiator, which then uses it as a forward source route to a router that makes progress toward the destination. If a router does not place its address into the router list, an invalid source route is usually constructed. This may result in an illegal routing entry being made by the flooding initiator. This implies that, for some set of destinations, the initiator will now route traffic over a route that does not make progress toward the destination. The effects of this were discussed on page 27.

Altering the Source Route

Any router along the source route, either in a flooding reply packet or in any source-routed data packet that passes through it, can alter that source route. The effect of this was discussed in previous paragraphs.

Failing to Forward Correctly

A router may not forward a flooding request properly. As long as the flooding limit f has not been exceeded, it is required to add its address to the flooding router list and transmit copies to all its neighbors except the one from which it received the request. If it does not transmit all required copies, it diminishes the region of the network that is searched to satisfy the flooding request. That could result in dropped connections.

Failing to Reply Correctly

A router may reply successfully to a flooding request when it cannot make progress toward the destination, or the terminating router may give a false address. However, this need not result in an incorrect routing entry in the initiator's routing database. In some cases, the bad router's immediate neighbor on the reverse source route can detect this and discard the packet. The initiator can also determine from the terminating address on the source route it receives whether or not progress is made. It is also possible for the initiator to detect alteration of the destination address that provoked the flooding request. If an invalid response passes these tests, then (since it receives a set of replies) the initiator may at times ignore invalid responses in favor of some other response.

3.2.3 Effects of Altering Data Packets

Data packets have several fields which must be unenciphered within a router. They are the source address, the destination address, the progress limit field, and the source route. Each can be illegally altered by a router.

Altering the Source Address

The *location* portion of the source address is used by the procedure to inform the other end of a connection of a change in the source's location. This is used by the other end to change its destination address and so to automatically reroute subsequent traffic to the correct location. Altering the source address incorrectly will result in dropped connections. The altered packet would proceed to the wrong location, would eventually be discarded, and would add to network overhead.

Altering the *identity* portion of the address will either cause a packet to be dropped by a destination gateway or cause it to be delivered to the wrong host. This could result in dropped connections and would add to network overhead.

Altering the Destination Address

The effects of altering the destination address are similar to those created by altering the source address.

Altering the Progress Limit Field

The progress limit is set in a packet after a routing decision is made. It contains the distance of closest approach to the destination's location that can be achieved, either by forwarding to a neighbor or by a source route obtained through limited flooding. If the limit field is altered, it could result in a loop. This can often be detected and avoided through examination by routers that check for consistency of the various fields against the progress criteria.

Altering the Source Route

The effects of altering the source route were discussed on page 27.

[illegible].4 Altering Field Values in Combination

An attack can result in combinations of the alterations discussed above. The most serious effect would be an undetected routing loop. As before, quite a number of these attacks could be detected by routers that check for consistency of the various fields against the progress criteria.

3.2.5 In Conclusion

The Cartesian routing procedure meets practical routing requirements. Unlike the ARPANET procedure, it places relatively few restrictions on router behavior. It is less efficient than the ARPANET procedure but far simpler to implement. The Cartesian procedure does not possess the behavioral characteristics that would cause a complete network failure if any single router were maliciously attacked. Therefore, it is more attack resistant than the ARPANET procedure. As will be shown later, the Cartesian procedure is amenable to techniques that further improve its attack resistance by allowing routers to detect invalid flooding data.

4 Mechanisms for Reducing Exposure to Damage

Routing procedures define and restrict legal router and link behavior. These restrictions vary for different routing procedures. Behavior that falls outside those restrictions can damage a network. At the very least, protocols should include a ***robustness principle***, such as that in IP and TCP, that allows a router to detect and deal with packets that have obviously been generated by incorrect procedures. Of course, this presupposes that routers can detect bad packets and handle them in a reasonable manner, which is not always possible. Ideally, we wish to define routing procedures so that illegal behavior at any one point cannot greatly damage an entire network (assuming that the network is topologically well connected). As a practical matter, it may not be possible for certain classes of routing algorithms to be made fail-safe in this way.

Any router or link can become the source of an invalid control message. Protection against a link's alteration or replay of control messages can be accomplished by a combination of point-to-point encryption and time-stamps or sequence numbers. Preventing a router from generating any invalid control messages requires perfect software, perfect hardware, and completely tamper-proof routers.

Perfect software is probably not an achievable objective for any practical application. There is some question as to whether program verification will ever be able to prove the correctness of any large software system. Since program verification presumes the existence of axioms that describe correct behavior, there always remains the philosophical question: Who verifies the axioms?

Perfect hardware is likewise unachievable. Correct software execution requires correct computer operation. It is always possible for the execution unit to jump to an incorrect program address or for memory to fail, due to a transient hardware problem. These events are not always detectable. This renders moot the question of whether or not perfect software could be created.

A tamper-proof router is also a practically unachievable objective. For economic reasons, routers may be placed at relatively insecure locations, for example, inside customer premises. Routers require servicing, replacement, and probably periodic software reloading. There will always be people who have access to routers and who must be trusted. In a commercial environment it is possible for someone with special knowledge of the router to maliciously attack the network. Recent events indicate it is probable that some one would try to attack a network merely to achieve personal gratification. In a military application, router sites could be attacked and overrun; interrupting enemy communications is a common military tactic.

For those reasons, it is wise to design network operating software so that it becomes much less likely for an attack at any single point to cause widespread network damage. To achieve this may require fundamental changes in network operating software. For example, it may not be possible to design a practical, attack-resistant shortest-path routing procedure.

4.1 Weakening the Requirement of Reliable Delivery

Much of the theoretical work on routing has concentrated upon shortest-path and related routing procedures, because they afford maximum or near-maximum reliability of delivery. However, shortest-path routing procedures require network-wide propagation of routing updates to achieve reliability while at the same time preventing loops; and hierarchic routing, while limiting the spread of updates across cluster boundaries of the hierarchy, still requires unlimited propagation throughout all levels. Such propagation may expose a network to partitioning if it is properly attacked.

Choosing Another Class of Routing Procedure

A commercial network can be designed with relatively few, large, centralized routers. This is the approach used in most telephone systems. Communications channels are assumed to be reliable, as are the routers. Routing in such a system is largely static. Routing changes when they occur are infrequent, and in some cases they require physical intervention. In a military network one must assume that neither communications channels nor routers are so reliable. Centralized, static routing procedures are not as desirable in a military communications network as they are in the telephone system.

Early networks were small enough to allow application of true shortest-path procedures, but network growth will soon force designers to abandon shortest-path routing and to relax reliability somewhat. If a new routing procedure is chosen, consideration should be given to procedures that do not require the network-wide propagation of routing updates, thus limiting the propagation of an invalid update and the extent of damage that may result.

4.2 Simplifying Routing Software

The complexity of the network operating software has a direct affect on the number of indirect attacks that can occur. It is an arduous task to list all restrictions that a suite of operating software either explicitly states or implicitly assumes. The number of possible indirect attacks is combinatorially related to the number of those restrictions. An examination of what might happen if any restriction is violated, by itself or in combination, is required to demonstrate attack resistance. Simplifying network operating software also diminishes the probability of an undetected programming error.

If the software is simple enough, it can be incorporated into the hardware. This not only increases the packet processing rate, but can make tampering much more difficult. Unfortunately, the complexity of the software programs running inside the routers of most networks, such as the ARPANET and Internet, makes that approach to tamper resistance impractical.

4.3 Link Encryption

A network becomes considerably more resistant to indirect attack when it enciphers the packets sent over the links between routers. Several classes of indirect attack can be made on a link. These involve selectively damaging or discarding packets, masquerading as one end of a connection, or inserting invalid network control messages. Those forms of attack require a filter either to examine the packets that pass through it, or to create and insert correctly constructed packets.

Link encryption prevents these attacks. Assume that the encryption algorithm is good and that the keys used by the routers at either end of a link are kept secret. If only link-level start/stop packet framing data remains unenciphered, then a filter could not examine packets for their source address, destination address, or packet contents (which is necessary if a filter is to selectively discard or damage traffic). Neither could a filter create packets that would be decipherable to a router at either end of the link. Therefore, it could not mimic a host or create network control messages. For additional protection, the beginning and ending position of a packet within the frame could be enciphered and made to vary randomly via padding, thus effectively hiding the size or start position of a packet. Unused bandwidth on a link could be consumed by 'junk' packets that would be immediately discarded when received. This would make traffic analysis quite difficult and would further complicate the task of selectively damaging traffic.

4.3 Virtual Circuit vs. Datagram Routing

It is not the purpose of this report to stress the differences between virtual circuit routing and datagram routing. To achieve attack resistance, one must ensure the validity and integrity of the routing information exchanged under virtual circuit routing procedures, just as under datagram routing procedures. Although these two classes are quite different, the attack-resistant datagram routing procedure developed later in this report can also be used to create virtual circuit paths.

Virtual circuit routing requires end-to-end path negotiation before a connection is established and data traffic can begin to flow. Because virtual circuit procedures are more complex than datagram procedures, one suspects that the task of achieving attack resistance for a virtual circuit procedure would be more complex. Under virtual circuit procedures, the strong tendancy to use only a single route at any given time makes it much easier for an attacker to completely intercept, alter, or block all traffic on any circuit that passes through it. Thus there is some indication that achieving attack resistance for virtual circuit procedures may be more difficult. The issue of the attack resistance of virtual circuit routing procedures will not be discussed in any detail in this report. It should be the subject of a future study.

4.4 Hop-by-Hop vs. End-to-End Acknowledgment

A network that employs hop-by-hop acknowledgment of packets is somewhat more resistant to indirect attacks than one that relies upon end-to-end acknowledgment.

Routers in a network that do not employ hop-by-hop acknowledgment determine whether or not a link is in operating condition through the periodic exchange of 'ping' packets with their neighbors. This may take the form of hop-by-hop acknowledgment of a particular class of packets, such as routing update packets. An attacker can then identify the class of packet utilized to assure link operation and allow them to pass, but selectively discard or damage other classes of traffic. This produces a situation in which a link appears to be correctly operating when it is not.

Hop-by-hop acknowledgment of all packets prevents the successful application of this type of attack. Consider an attack made on a link that involves the discarding of packets. If the router is responsible for retransmission and acknowledgment of all packets, then any attack that consistently discards or damages any particular packet will eventually cause the transmitting router to declare the link inoperative and trigger network rerouting. Without hop-by-hop acknowledgment the point of attack is known only to be somewhere along the route from source to destination.

Two drawbacks to hop-by-hop acknowledgment are the additional overhead in both processing time and bandwidth. At some point, as the transmission time along the link increases, hop-by-hop acknowledgment becomes impractical. Geosynchronous satellite communications links exhibit round-trip times on the order of one-third second. In such a case, the massive buffering required to efficiently utilize bandwidth makes hop-by-hop acknowledgment impractical. An additional objection is that hop-by-hop acknowledgment is usually implemented because it discards damaged packets and requests retransmission much more rapidly than end-to-end acknowledgment. While this would normally be considered a desirable feature, in certain situations, real-time data that is delivered damaged but promptly is still useful, whereas late but undamaged data is useless. This problem can be avoided by the addition of a packet header option.

Hop-by-hop acknowledgment does not protect a network from all attacks that involve the selective discarding of packets. It is designed to detect failed *links* rather than failed routers. Just as a router could discard any packets it received while simultaneously acknowledging them, so could a filter placed on a link. If this form of indirect attack is consistent, then the evidence for an attack would be a dropped connection and the point of attack known only to be somewhere along the route from source to destination. If only one is to be used, then link encryption is preferable as a method for increasing attack resistance, since it not only avoids the overhead associated with hop-by-hop acknowledgment but also prevents a filter from successfully acknowledging packets.

4.6 Weakening the Requirement of Flexibility

The requirement of topological flexibility may not be necessary for a commercial computer network, since in the future it will be possible to replace telephone exchanges with extremely large and complex packet routers. This is the approach being studied by various telephony organizations such as AT&T [Turner-86]. Since central exchanges are rarely added and are almost never removed, the need for topological flexibility in the network core decreases. Flexibility primarily needs to be retained at the edges of the network, where customers are connected.

This presupposes that future wide-area commercial networks would be built on top of the current telephone network. It should come as no surprise that telephone and telegraph organizations in many countries are the principal supporters of the ISDN development effort. If development actually proceeds in a less centralized, more distributed manner, the need for topological flexibility grows. The actual path of development will depend upon patterns of demand, cost, and the regulatory environment. An alternative to telephone system physical distribution is now available in many metropolitan areas via cable-television systems. It is unclear whether or not CATV systems could eventually provide high-speed two-way communication at commercially acceptable rates. It is technically possible for CATV systems to provide that service. A first step in that direction would band-isolate two-way communication traffic from the predominantly downstream, broadcast CATV traffic. That would allow relaxing the CATV topological restriction from that of a rooted tree, to a generalized cyclic graph for those bands carrying two-way traffic.

Because centralization leaves a network vulnerable to physical attack, it is undesirable for a military network. Furthermore, by its nature the military requires routing procedures that allow a very high degree of topological flexibility. Communications must move with personnel; routers would be moved, added, and removed from differing geographic areas of the network as conditions changed.

4.7 Limiting Attacks that Make Use of Priority or Broadcasting

Networks that allow multicasting (or broadcasting) expose themselves to an attack that takes the form of excessive transmission of packets that request broadcast or multicast distribution. When that is allowed, it is sometimes possible for a single router to generate enough multicast packets to overload the other network routers. This will happen if the multicast message requires an abnormally large amount of processing on the part of the receiver; an example is a network-wide routing update distributed via flooding.

If the set of routers participating in multicast groups is kept small, it becomes substantially more difficult to overload the network. However, it remains relatively easy to overload the hosts within a group. For example, each workstation on a LAN that receives a broadcast message must examine that message to determine whether or not any action is required of it. Normally, at least two context switches occur within the receiver's processor for each broadcast packet received. The denial of service that this implies, for both network and host populations, suggests that mechanisms be implemented to limit the possible abuse of broadcasting.

We distinguish broadcast messages generated by routers as a part of network operation from those generated by hosts. Routers should limit the rate at which hosts originate broadcast messages. If possible, they should also enforce this limit on their neighbors by discarding any excessive broadcast traffic they might receive.

Depending upon the particular routing procedure, it may not be possible to apply this restriction to broadcast messages that are generated by routers. For example, in the ARPANET procedure, updates that arrive at a router must be quickly redistributed by flooding. The short-term arrival rate of routing updates is statistically distributed and essentially uncontrollable; it is illegal to queue them to 'smooth' the rate at which they are redistributed, and also illegal to discard a valid update without first reforwarding it. However, there is a maximum rate that governs the generation of new updates by any single router. If an immediate neighbor notices that the rate is being exceeded, action could be taken to limit the distribution of the offending router's updates.

4.8 Segregating Networks by Trust

It has been observed that the short-haul and local-area portions of networks are less stable than the long-haul portions. As a rule of thumb, additions and modifications to internetwork topologies occur much more often on LANs and LAN clusters than on portions of the internetwork that terminate long-haul communication links. Long-haul links can be presumed to be stable for the following reasons:

- They are relatively expensive.
- Adding a link or modifying its physical routing requires a great deal of lead time.
- They are essential to the total operation of the network.

Administrative benefits are gained by using stable hardware and well-debugged code to operate the long-haul portions of an internetwork. At the 'edges' of the internetwork one observes greater variability in hardware and software. In this situation the long-haul, 'central' portion of the internetwork is more trustworthy than the edge portion. This suggests that some attack resistance could be gained by placing a logical firewall between the center and the edges of a large internetwork. The Internet is now in the process of doing this.

Planned Internet Modifications

As the Internet has grown, a number of routing problems have arisen:

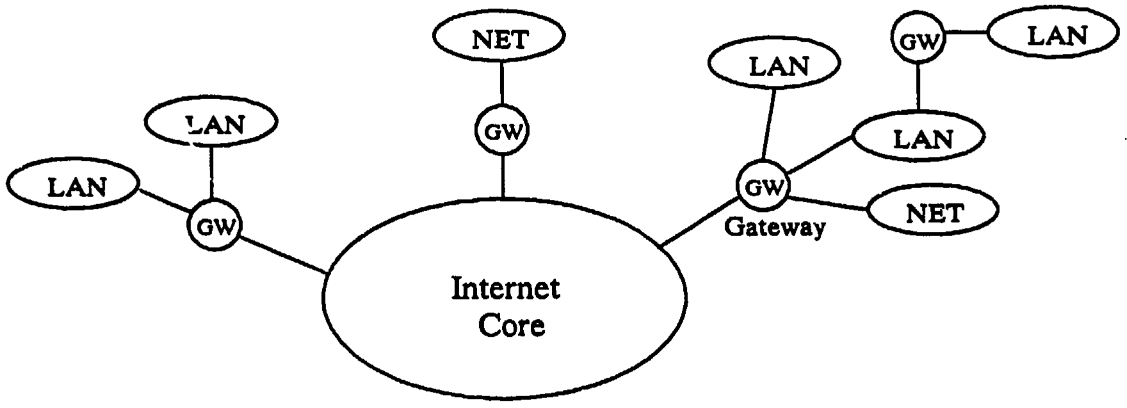
- Overhead grows too fast. This includes growth in routing table size, update processing time, the number of update messages, and network bandwidth consumed by update messages. All of these grow approximately linearly with the number of routers.
- The proliferation of dissimilar physical gateways, sometimes with insufficiently tested routing software, has led to difficulties such as

routing loops. The rapid, network-wide distribution of routing data has made fault isolation difficult.

- The Internet gateway routing software itself is complex. It is extremely difficult for network administrators to distribute changes to the various types of gateways and to see that they are installed correctly.

Researchers have proposed a three-level routing hierarchy that attempts to address these problems. At the center of the Internet remain the *core* gateways, which still run the Internet routing procedure, or soon a variant of the ARPANET's SPF procedure. They constitute a smaller manageable tier, which is responsible for long-haul network communication. At the edges, where local area networks are connected and where most of the growth occurs, are located the exterior gateways. These run an *Interior Gateway Protocol* (IGP). Several IGP's may exist, each perhaps with its own routing procedures. IGPs exchange routing data with one another and the core gateways via an *Exterior Gateway Protocol* (EGP) [Rosen-82][Seamonson-84][Mills-84].

Figure 5.



The EGP restricts the network topological model to a tree that is rooted to the Internet core system (see Figure 5). In theory this prevents the newer, more diverse population of routers from causing routing loops that might affect the entire Internet. An errant gateway can affect only those routers in the same rooted branch as itself. The core is highly trusted and runs a shortest-path routing procedure that allows a more general topology. Unfortunately, nothing in this proposed hierarchy prevents LANs in separate branches from installing a gateway between themselves and so violating the model. This could result in routing loops.

The topological restrictions inherent in a tree will eventually become bothersome. Further divisions are proposed that would aggregate like gateways into *autonomous systems* [Mills-86]. The gateways in an autonomous system must be able to reach one another via paths that do not leave that system. The routing topology within an autonomous system is not restricted to a tree. One or more autonomous systems that trust one another and share a common security model can be further grouped into an *autonomous confederation*.

As before, gateways in an autonomous confederation must all be able to reach one another without leaving that confederation. The core system would then be just one more autonomous confederation. From the perspective of the ANSI/ISO routing proposal mentioned earlier, one could view autonomous systems as similar to Domains, and autonomous confederations as Dominions.

4.9 The Use of Authentication Servers

The possibility of spoofing or replay attacks involving routing messages has recently become a subject for discussion in the literature. How does a recipient of routing information know both that it was generated by an authorized source and that the information has not been modified en route? By using a trusted third party as an authentication server, it is possible both to authenticate the source and to provide data integrity of messages sent by the source to specific destinations. However, authentication servers do not validate the data placed into messages, so the possibility remains that invalid data can be sent without detection.

One proposal suggests using an authentication server to restrict participation in the EGP protocol to only those gateways known to be participants in the protocol, thus ensuring the integrity of data exchanges between them [Mills-87]. Two authorized EGP-gateways that desire to exchange routing data would first obtain a session key, which they would then use to encrypt the checksum and sequence numbers of the update messages they exchange. Gateways would be able to determine whether or not an update packet had been altered while en route between them. They could also detect false EGP messages sent by some other host or gateway masquerading as an EGP-gateway.

An authentication server is a trusted network resource that contains passwords for each authorized user of a particular service. If greater network reliability is to result from their use, authentication servers must be more reliable and secure from attack than the average network host. Networks should be subdivided into fully overlapping service regions, each with its own primary authentication server. This allows each network router to consult at least two authentication servers. If a primary becomes inoperative or unreachable, then a back up will allow operations to continue in the affected region. Furthermore, in a large network, servers would only be required to store passwords for hosts in their service region. This solves a problem of scaling and limits the number of hosts compromised if a server's password file is siezed.

The increasing use of servers in networks for performing vital tasks, such as name-to-address mapping, was discussed earlier. These information servers differ from an authentication server, since they respond directly to requests rather than acting as a mutually trusted third party. However, the data they return to the requestor must be protected from modification while en route. This implies the use of a password file. The authentication servers are logical candidates for the location of these other services, since they already contain the needed password file. Additionally, these other services would share the greater physical security that presumably surrounds the authentication servers.

Since a server's data is not validated, access to its database must be carefully controlled and the data in it carefully examined before modification.

4.10 Developing Multiple Simultaneous Paths to a Destination

One requirement of shortest-path procedures, and also of virtual circuit procedures, is that a router can use only one path to a particular destination at a time. There always remains the possibility of a malicious router's intercepting, discarding, or altering the traffic that passes through it. If simultaneous multiple paths are maintained by a router to each destination, then, by a combination of random path selection and end-to-end retransmission, the probability of a connection being severed by any single indirect attack can be diminished. A routing procedure that allows the simultaneous use of multiple paths is inherently more attack-resistant than one that does not. The Cartesian procedure has this characteristic and therefore has a distinct attack-resistant advantage.

5 Basic Requirements for the Attack Resistant Exchange of Routing Information

Some degree of attack resistance can be added to routing procedures through the use of point-to-point link encryption, but ideally a much stronger resistance to attack is desired. Procedures are required that:

- detect network control messages containing damaging and invalid data, and prevent or correct the damage that might occur from their distribution.
- incorporate no software behavioral restrictions that, if violated by a single router or host, could greatly damage an entire network.
- satisfy the routing requirements of reliability, flexibility, and efficiency.

Any router has control over the composition of messages that pass through it or that it constructs. Any message can contain invalid data, placed there either by the source or by any intermediary between the source and the recipient of the message. To determine whether or not a routing update message is potentially damaging, it must be possible for the recipient to detect invalid data of sufficient severity, or to know that some other authority validated the message and that no intermediary between the source and the recipient modified that message.

For the recipient to detect all invalid, damaging data requires that it be omniscient. For example, for a recipient to determine that a topological update (claiming to have a low-delay path to some other network router) is erroneous requires the recipient to know about the network topology of the region mentioned in the update. If updates are distributed throughout the network, then each router must possess knowledge of the entire network topology. This is impractical for large-scale networks.

5.1 Detecting Modification of Routing Messages

Preventing a message from being modified undetectably by an intermediary is usually accomplished via encryption. Sensitive data is enciphered at the source with a key that is known only by the intended recipient. Only the source can undetectably modify the message, and only the intended recipient can correctly decipher the message's data. However, applying encryption to routing update messages can also lead to problems. To prevent undetected modification by intermediary routers, encyption keys must be unique for each [source,destination] router pair. For N hosts, this implies the necessity of $O(N^2)$ unique keys, since each router would need to store N-1 keys. Whenever a router is added or removed, every router must be modified. Unless a network is small, it is not practical to protect network control messages by this static storage of keys.

Two solutions to the static storage of keys generate them dynamically upon request. Assume that a regional authentication server AS exists, that it possesses private keys for each router in its region, that both routers **A** and **B** are within that region, and that **A** wishes to send an update *r* to **B**. **A** builds a message in which it identifies itself, and encrypts both the address of the destination **B** and the data *r* using its private key. It then transmits this to **AS**, which decrypts the message using **A**'s private key. **AS** then extracts the destination **B** and builds a message encrypting *r* using **B**'s private key. **AS** then transmits that message to **B** along with a time-stamp that contains the time when **AS** received **A**'s message.

$$A \rightarrow AS: \quad A, \{B, r\}^{KA}$$
$$AS \rightarrow B: \quad AS, \{r, t\}^{KB}$$

A modification to that approach uses **AS** as trusted third party to generate a session key K_S used by **A** to communicate with **B** [Needham-78]. **A** sends a message to the authentication server **AS** that identifies itself and the destination **B**, and contains a once-only transaction identifier I_A, which is used to prevent replay of any previous response from **AS**.

$$A \rightarrow AS: \quad A, B, I_A$$
$$AS \rightarrow A: \quad \{ I_A, B, K_S, \{K_S, A, t\}^{KB} \}^{KA}$$
$$A \rightarrow B: \quad \{K_S, A, t\}^{KB} \quad \{r\}^{K_S}$$

In both approaches the possibility of spoofing must be curtailed. A relatively inexpensive way to manage this threat is to have the source include a time-stamp value *t* as part of the data it returns. If network clocks are roughly synchronized, this time-stamp can be used to prevent replays where the difference between the destination's clock and the time-stamp is greater than some limit $a+\Delta t$ (where Δt is the maximum allowed synchronization error).

From the standpoint of validation, the first approach to static storage requires that *r* be validated by **AS** and **B**, while the second requires that it be validated by **B** and any intermediaries on the path from **A** to **B**. Under each approach, only the authentication server needs to know the keys. However, both approaches have drawbacks that could prevent their use in large networks. Use of an authentication server slows the propagation of control messages, since it requires a round trip to and from **AS** before a control message can be sent. The amount of added delay depends upon the distance between each router and its nearby authentication server. A less serious problem is that a new router could not be added until the appropriate authentication servers were notified of its existence.

The delay problem is exacerbated if flooding is used as the mechanism to distribute update messages. An $O(N)$ computation cost arises then, since flooding must now be replaced by serial, repeated transmission. This greatly extends the period between the time an update is transmitted to the first recipient and the time it is received by the last recipient, and may in fact make this type of encryption impractical for use by routing procedures that rely upon a limited maximum update reception time.

It appears that the use of an authentication server can be avoided by using the signature authentication procedure available through public-key encryption [Rivest-78]. In a public-key system, each router A publicly discloses an encryption procedure E_A. It keeps private an associated procedure D_A. If you wish to transmit a secure message M to A, you first encrypt it with A's public key, producing $E_A(M)$. Upon reception, A decrypts using its private procedure $D_A(E_A(M))=M$, resulting in the original message M. If signatures are allowed, then procedures D_A and E_A permute. If A wishes to transmit a secure message to **B** (and if it is important that **B** knows it in fact came from A) then **A** uses the following procedure. It encrypts the message using its private procedure, producing $D_A(M)$. This is transmitted to **B**, which decrypts using A's public procedure $E_A(D_A(M))=M$, reproducing M.

If a public-key system is used, then N^2 unique keys are no longer required, but the other drawbacks remain. Potentially, each router must be able to access a public key for all other routers in the network. If N is large, it is impractical to require routers themselves to store the keys. Authentication servers would be required to reliably store and distribute them. As before, if a message is distributed to all routers, an $O(N)$ computation cost arises, since $D_i(M)$ is a unique computation for all routers i.

5.1.1 Simultaneous Requirements of Validation and Data Integrity

Traditional end-to-end encryption techniques solve the problems of data exposure and data modification. This is sufficient to protect messages against modification by an intermediary. However, any message may contain invalid data inserted at the source. That situation must also be detected. In the case of routing updates, any update containing damaging data should be detected and removed before damage is done to the network. This requires that some outside authority examine the message and validate its contents. If a router builds an update, that update must be validated either before or upon arrival at its destination.

There are two approaches to the problem of validation:

1. Use a regional authority such as an authentication server to validate a message.
2. Use routers, other than the one that built the message, to validate the pieces of the message about which they possess knowledge, as the message travels to its recipient.

The first approach places complete trust in the hands of some few regional authorities. All data necessary for validation is stored there. Since validation is performed before the message is sent to its recipient, there is no requirement that the update data be readable by any intermediary. The second approach appears considerably more complex.

Under the second approach, necessary validation data is assumed to be distributed piecemeal throughout the network. It is not clear that this is either possible or practical. Any router that performs validation must be able to read the message, implying either that it can decipher the message or that the data is in the clear. But if the data in the message is ever entirely in the clear, then the validator can alter it undetectably. If only the destination performs the validation, this implies that all routers possess the necessary data to perform a validation. When validating a routing update under these constraints, reliability and topological flexibility become impractical for all but small networks. A mechanism is required that allows control message data to remain in plain-text while detecting any modification of that data.

A data checksum is the usual mechanism employed to detect data alteration during transit or storage. However, each router presumably knows the network's data checksum algorithm and can generate the correct checksum value from any data field. Therefore, any intermediary router can undetectably alter message data if the only way to detect such modification is the checksum.

A possible solution would be to create a second checksum by duplicating and enciphering the data checksum with a session key. The resulting integrity-check value would be similar to a data checksum, except that intermediaries would not possess the [source, destination] encryption key used to encipher it. The destination would receive the message data, calculate a checksum value based on that data, decipher the integrity-check to derive the original checksum value, and compare the two. If an intermediary had altered the message data, would those two values match?

Network checksum algorithms are many-to-one valued functions. Many modifications of the checksummed data are possible that result in the same checksum value. Checksum algorithms are designed to detect *stochastic* modification of data. An implicit assumption is that the data has not been attacked in an intelligent, planned manner. For example, the Internet IP/UDP checksum algorithm is able to detect stochastic data modification but a simple swapping of any two aligned 16-bit words of checksummed data results in the same checksum value. Other checksum algorithms are better in the sense that it is more difficult to generate identical checksum values from a reordering of the input data. It nevertheless remains the case that if an intermediary can compute the checksum value, then any many-to-one checksum algorithm allows that intermediary to indetectably alter the data.

For the source to simultaneously provide the data in both clear and enciphered forms largely solves this problem. Virtually all modifications to the clear data by an intermediary would be detectable. The destination would use the session key to decipher the enciphered copy and compare it against the clear copy provided by the source. This technique in effect creates an additional checksum value of equal size to the data. The mapping between checksummed data and this checksum value is now one-to-one instead of many-to-one valued. The original checksum can still be used by intermediaries to detect data errors during transit.

This additional protection is gained at the expense of bandwidth and processing time. Duplicate data is transmitted, and the destination must decipher and compare data copies. Still, the effect of the additional bandwidth should be small, since control messages are infrequent relative to user data messages under normal operating conditions in a well-used network.

5.2 Validation of Source Routes

In a large network it is impractical for any single authority to store a complete topological map that would allow it to validate all routing information exchanges. To be practical, validation information should be distributed among regional authorities. One way to make validation practical is to restrict the distribution of any particular routing update to a restricted population of routers, so that regional authorities may perform validation. Another way is to create a mechanism that allows piecemeal validation of an update as it proceeds toward its destination. These restrictions are met by a routing procedure that transmits routing information from one router to another in source route segments.

Let the neighbors within one hop of router i be represented by the set $N_{1i} = \{r \mid r \text{ is directly connected to } i\}$. Assume that a router communicates information concerning connections to its immediate neighbors. In Figure 6a, x sends such an update to its neighbor w, claiming that x is connected to v. Without an outside source of information, w has no way of determining the validity of x's assertion; v may or may not be a member of N_{1x}, or v itself may not exist. This fundamental weakness allows attacks that produce partitioning.

Let $N_{2u} = \{N_{1i} \mid i \text{ is directly connected to } u\}$ be a set containing, for each of router u's neighbors, the set of their neighbors. Assume that sets N_{2u} are correct and that each router u possesses this set. In Figure 6b, let N_{1x} = {w, y, and z} and assume that x falsely claims in an update that it has a link to v. Since w possesses in N_{2w} a validated N_{1x}, w can immediately determine that x's claim is invalid.

Figure 6a.

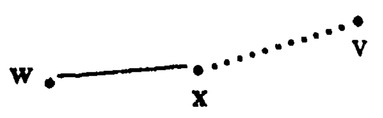
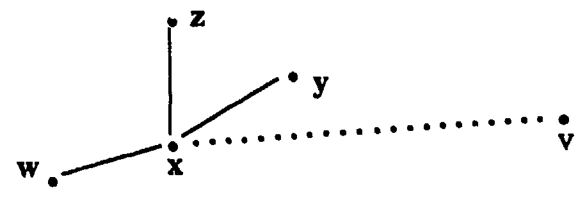


Figure 6b.



Consider a source route $\{x_1, x_2, \ldots, x_{n-1}, x_n\}$ communicated by x_{n-1} to x_1. By induction, the entire source route can be validated hop by hop as the packet containing the source route proceeds toward x_1. Each router x_i, i=1, ..., n–2 can verify the addresses claimed and can also verify the existence of links from itself to x_{i+1} and thence to x_{i+2}. This form of source-route validation does not require the use of an outside validation authority, but it does assume that a source route is not modified during transit through the various routers x_i. To provide data integrity it will still be necessary to obtain unique [source, destination] keys for x_1 and x_{n-1} from an authentication server, so that an enciphered copy of the source route is also available for comparison by x_1.

This neighbor validation mechanism allows a router to detect a fabricated communications link or router. One drawback is that it provides no information concerning the characteristics of any link, or any delays associated with it. Another is that neighbor validation restricts topological flexibility. It is assumed above that the databases associated with N_{2u} are correct. This implies that a link could not be added to the network without the neighbors of the routers at either end of the new link first being informed of the addition by some trusted outside administrative authority. If a link is removed, neighbors at either end must likewise be informed. This mechanism makes maintenance activity somewhat more difficult. If validation is performed instead by a regional validation authority, much the same restriction applies; the regional authorities must be informed prior to any semi-permanent topological change.

5.2.1 Applicability of Neighbor Validation

Can the technique just described be applied to commonly used routing procedures? In the distribution of routing updates, the minimum information distributed is address and topology. (This is what the current Internet procedure distributes.) However, under that procedure a router distributes topology information for sites other than its immediate neighbors. Thus it is unsuitable for the application of neighbor validation. The same seems true for the hierarchic procedures.

Under the ARPANET procedure, a router limits the information it distributes to the address and delay data of its immediate neighbors. Unfortunately, the routing procedure distributes it across the entire network via flooding. To ensure integrity of the data via secret [source, destination] keys requires an increase in overhead and distribution delay proportional to the number of routers in the network. It is not practical to apply neighbor validation to the ARPANET procedure.

The Cartesian routing procedure constructs source routes. An initiating router requests a route that is constructed hop-by-hop between itself and some terminating router. Each source route can be validated using neighbor validation. Each source route is sent from its terminating router back to its initiator using the source route in reverse. It is possible to simultaneously apply neighbor validation and to ensure integrity of the data via secret [source, destination] keys without an increase in delay proportional to network size.

5.2.2 Distribution of Authentication Servers, Their Storage Requirements, and Overhead

The use of regional authentication servers has been postulated to ensure the integrity of transmitted routing information. Each server is required to store the private keys of the routers in its region. It is assumed that the network administration ensures that a sufficient level of physical security surrounds each authentication server. How must these servers be distributed throughout a network and how much storage do they require?

Assume that no source route will ever exceed a length of l. Consider a source route $\{x_1, x_2, \dots, x_{n-1}, x_n\}$, and assume that x_{n-1} requests its regional authentication server AS to validate that route. If AS alone is to provide the private keys of both routers, then x_1 and x_{n-1} must be known to AS. If authentication servers are distributed so that no router is farther than m hops from the nearest server, then each server must store keys for those routers within (l+m−2) hops of its location, since x_{n-1} cannot be more than m hops from AS, and x_1 cannot be more than (l−2) hops from x_{n-1}.

Assume that the network has a grid topology with routers equivalently spaced along both axes. The address and private key storage requirements for an authentication server are equivalent to the number of routers within (l+m) hops of the server, or $O(2(l+m)^2)$.

(l+m)	$2\,(l+m)^2$
10	200
20	800
40	3200
80	12800
160	51200
320	204800

As the table demonstrates, storage requirements increase slowly enough that a single authentication server could easily store the passwords of all the routers within 320 hops distance. However, the administrative overhead to maintain the table for some 205,000 routers would probably make table maintenance impractical, since each private key change or addition/removal of a router within the sender's 320-hop region requires a change to the table. If the regional network administrator can securely update each authentication server's tables remotely, then the overhead associated with key changes and the addition/removal of routers should be manageable for servers that cover substantial regions.

It is unlikely that source routes of length l=20 would be used even under very adverse circumstances. For example, if one were to use Cartesian routing in the ARPANET, a flooding limit no larger than f=4 is required under normal circumstances, which implies a lower operating bound of l=6 [Finn-87]. If a network were to use piecewise source routing, the network's administrator would almost certainly alter basic network topology so that long source routes would not be needed under typical conditions.

More important will be the service rate that must be maintained in order to provide timely response to requests for session keys, to validate data, and to transmit its responses. A delaying factor is the number of intermediary routers that lie between the requestor and the server. A balance must be struck between these two sources of overhead and the cost of additional servers. It would also be necessary to provide at least a single level of server redundancy to retain attack-resistant characteristics.

5.2.3 Subdividing the Authentication Task

Is there a practical way to share validation information for source routes of extreme length among a group of validation authorities? If each authentication server maintains source routes to neighboring servers and knows their private keys, then it is possible to validate a source route hop by hop. This would allow long source routes to be validated, but it would take longer to do so.

Assume that authentication servers are distributed so that at least one server is within **m** hops of any router, and that the servers know the keys of all routers within **m** hops of their location. Assume also that each server maintains source routes to its neighboring servers within a radius of at least 2**m** hops. An authentication server **AS** receives a request for private keys for routers x_1 and x_3 associated with a valid source route $\{x_1, x_2, x_3\}$.

Router x_2 is within m hops of **AS**, which contains x_2's address and private key. But since x_1 can be m+1 hops distant, **AS** may not know x_1's private key. If x_1 exists and **AS** does not know its key, then some neighboring authentication server must contain that information. For each of its neighboring servers, **AS** builds a packet that requests the private key for x_1. It then enciphers the packet with each neighbor's private key and transmits these packets to them using the source routes it maintains.

$$AS \rightarrow AS': \ AS, \{x_1\}^{K_{AS'}}$$

$$AS' \rightarrow AS: \ AS', \{x_1, K_{x_1}\}^{K_{AS}}$$

Each authentication server that receives such a packet decrypts it using its private key. If the neighboring server does not contain information concerning x_1, it discards the packet. If it does contain the requested information, it builds a reply packet that contains x_1's private key, enciphers the packet with AS's private key, and transmits the packet back to AS. This procedure allows AS to complete its task.

Assume now that servers are regularly spaced across the network with **m** hops between them. Now consider a longer valid source route $\{x_1, x_2, \ldots, x_{n-1}, x_n\}$. Since x_{n-1} is within **m** hops of some server **AS**, then as long as $m+2 \geq n$, x_1 must be within **m** hops of **AS** or one of **AS**'s neighboring servers. The above search procedure now finds keys for any x_1 that is part of a valid source route $\{x_1, x_2, \ldots, x_{n-1}, x_n\}$, where $m+2 \geq n$.

The search procedure can be generalized to search for the requested information in authentication servers even farther away, for longer valid source routes, via a procedure similar to limited flooding enquiry. This allows the storage requirements for an authentication server to be reduced from $O(2(l+m)^2)$ to $O(2m^2)$. Considerable additional overhead and time is required to validate long source routes.

6 Reducing Vulnerability in Cartesian Routing

The Cartesian routing procedure can be modified to employ both neighbor validation and data integrity. In the following sections the reader is assumed to be familiar with the basic operation of the Cartesian routing procedure [Finn-87].

Three classes of source routes are necessary to the operation of a Cartesian network; and they are:

(1) Source routes to at least two regional authentication servers.
(2) Source routes to at least two regional routers that reside in level $j+1$ in an m-level Cartesian network, for routers residing in an uppermost level j, $j=0, \ldots, m-2$.
(3) Source routes obtained after initialization and returned as the result of a limited flooding operation.

These routes must be constructed as part of router initialization. The addresses of the regional authentication servers and of the higher level routers, mentioned in classes one and two, are distributed by a network administrative authority in a secure manner to all concerned routers. These address assignments are relatively permanent. The sets N_{2i} are similarly distributed. Each router is assumed to have received all of this information prior to its initialization. Each router is assumed to keep a unique, private key, known only to itself and regional to authentication servers; the authentication servers themselves are assumed to be secure.

During initialization each router must obtain source routes to its regional authentication servers. If the router is part of a multi-level Cartesian network, it must also obtain source routes to regional routers in the network level immediately above it. These source routes are determined by a variant of the limited flooding procedure, in which the progress limit field is set to zero. This ensures that no valid flooding response can arrive from any address other than the specific address requested. Responses that do arrive contain source routes to the requested authentication server or higher level router. The validation and integrity mechanisms are discussed in detail below.

6.1 Achieving Attack Resistance when Using Neighbor Validation

In a single-level Cartesian network a router chooses the next-hop destination for a packet in essentially one of three ways:

(1) An immediate neighbor is chosen that is closer to the packet's destination.
(2) The router's database contains a source route segment that terminates closer to the destination. This new source route segment will determine the next hop.
(3) A limited flooding enquiry attempts to find routers within f+1 hops that are closer to the destination. If successful, the reply is a source

route that allows step (2) to be applied. If no reply is received, the destination may be assumed to be unreachable.

Every router knows about the existence and status of its immediate neighbors. It is desirable to validate all source routes when they are initially constructed, since these constitute the remainder of the paths used.

In section 3.2.2 of this report it was shown that an attack made by a single router on the flooding procedure during distribution of an enquiry has a minor effect. Because of the nature of flooding, some flooding enquiry packet will probably arrive at a router that can build and transmit a valid response. For the same reason, valid responses return to the initiator over a variety of paths. Unless the attacking router is on a critical path or there can be no valid responses, it is highly likely that a valid response will arrive at the initiator.

Assume that router x_1 requires a source route that makes progress toward location d (is closer to d than x_1), that it initiates a limited flooding enquiry, and that it receives $\{x_1, x_2, \ldots, x_{n-1}, x_n \mid d\}$ as a flooding response. The response claims that the router at location x_n makes progress and provides a source route to it. Ideally, it should be impossible for any false information in a flooding response to reach the initiator of the flooding enquiry and to be believed. Although this is not feasible, it is possible to make the procedure attack resistant.

False information can be supplied to x_1 by the router that constructs the response, in this case x_{n-1}; by any router $\{x_2, \ldots, x_{n-2}\}$ along the return path; or by any link along the route. Without additions to the network to improve security, there are some things x_1 can do for itself. The truth of the claim that x_n makes progress toward d can be immediately determined by x_1. However, x_1 may receive responses that seem to make progress, but that are false.

A response provides a source route to the initiator. If we assume that any data in the response may be false, then any of the routers mentioned in the route $\{x_1, x_2, \ldots, x_{n-1}, x_n\}$ may not exist, or the connections $x_i \leftrightarrow x_{i+1}$ between them may not exist. The initiator x_1 cannot by itself validate the entire source route. Router x_1 does have knowledge of its immediate neighbors, so it can determine if it is connected to x_2; but cannot know whether the remainder $\{x_3, \ldots, x_{n-1}, x_n\}$ constitutes a valid path.

Assume that during a flooding reply each router $\{x_i \mid i=1, \ldots, n-2\}$ determines that $\{x_{i+1}, x_{i+2}\}$ exist. Assume also that $\{x_i \mid i=2, \ldots, n-1\}$ make the same determination for x_{i-1}. If the conditions are not met, the source route is bad and the reply cannot be forwarded to x_1, and it is discarded. However, x_i could modify any portion of the source route $\{x_{i+2}, \ldots, x_n\}$ and this would not be detected by x_{i-1}. Preventing modification requires that the integrity of the data in any flooding reply must be ensured.

6.1.1 Providing Data Integrity for Flooding Replies

Data integrity can be provided, but it requires unique [source, destination] keys. It is not practical, nor is it wise for a router to store all the keys it might possibly need.

However, it is exceedingly unlikely that any single router would require even a small fraction of those keys. For practical network topologies, Cartesian routing limits the number of source routes that a router uses to a small integer, but just which source routes are needed is determined dynamically. This suggests that unique [source, destination] keys could be created with the aid of distributed authentication servers.

Assume that router x_{n-1} is responding positively to a flooding enquiry from x_1. Router x_{n-1} wishes to send a response to x_1 and to preserve the data integrity of its response. Following Needham and Schroeder's model [Needham-78], x_{n-1} receives from a nearby authentication server, AS, a session key K_S that it can use when sending its flooding response to x_1, and a time-stamp t.

$$
\begin{array}{ll}
x_{n-1} \rightarrow AS: & x_{n-1},\ x_1,\ I_{A1} \\
AS \rightarrow x_{n-1}: & \{ I_{A1}, x_{n-1}, K_S, \{ K_S,\ x_{n-1},\ t \}^{Kx_1} \}^{Kx_{n-1}} \\
x_{n-1} \rightarrow x_1: & \{ K_S, x_{n-1}, t \}^{Kx_1},\ \{ r \},\ \{ r \}^{K_S}
\end{array}
$$

Router x_{n-1} sends a response to x_1 containing three fields: (1) a field, encrypted by AS using x_1's key, which contains the session key K_S, x_{n-1}'s address, and the time-stamp t returned by AS; (2) the response r in plain text; and (3) the response r encrypted by x_{n-1} using K_S. Fields (2) and (3) allow x_1 to determine whether or not r has been altered en route. If x_{n-1} does not receive a response from an authentication server within a reasonable time, it does not respond to x_1 and attempts to find another active authentication server.

Note that both the plain text and enciphered versions of r are provided. This allows a known plain text attack to determine the session key. However, exposure of the session key provides little if any immediate advantage to an attacker. First, the session key is not used to hide any information, but only to provide an integrity check on the plain text. Second, a set of new session keys is generated for each individual flooding request. Finally, the initiator of a flooding request associates a timer with each request. If that timer expires, all remaining responses to that request are discarded. However, if K_S becomes exposed, one portion of field (1) enciphered with x_1's private key becomes known. It is still important that the encryption algorithm make a known plain text attack difficult.

In response to a limited flooding request, the message

$$\{ K_S, x_{n-1}, t \}^{Kx_1},\ \{ r \},\ \{ r \}^{K_S}$$

is sent toward x_1 along the source route r, represented by $\{x_1, x_2, \ldots, x_{n-1}, x_n \mid d\}$, in which each of the x_i are unique. The following steps occur before any reply to a limited flooding request is accepted.

(1) Consider $\{x_i \mid i=2, \ldots, n-2\}$. By using neighbor-validity, x_i can determine whether x_{i+2} is in fact a neighbor of x_{i+1}, and that x_{i-1}

is a neighbor. If it is not, the reply message is discarded; otherwise it is forwarded to x_{i-1}. By induction on i, intermediaries validate the entire source route r before it reaches x_1.

(2) Each router $\{x_i \mid i=2, \ldots, n-1\}$ examines the source route to ensure that all routers in it appear only once. If that check fails, the message is discarded. Each router x_i must receive the message containing the source route over the link from $x_i \leftrightarrow x_{i+1}$. If it has not, the message is discarded.

(3) Source routes of the form $\{x_1, x_2, \ldots, x_{n-1}, x_n, x_{n+1}, \ldots, x_{n+k} \mid d\}$ pass the above validation procedure, but $x_{n+1}, \ldots, x_{n+k}$ may be fabricated. However, the definition of limited flooding restricts valid source routes to be of the form $\{x_1, x_2, \ldots, x_{n-1}, x_n \mid d\}$. One of the fields communicated by **AS** to x_{n-1}, included in the packet containing the source route, is $\{K_S, x_{n-1}, t\}$, and it is encrypted using x_1's private key. Upon receipt x_1 decrypts that field, yielding K_S, x_{n-1}, and t. By inspection, x_1 can then determine whether or not the received source route is of the correct form. If it is not, the message is discarded.

(4) Router x_1 decrypts the enciphered copy of r using K_S. It compares the deciphered and clear copies of r. If they do not match identically, the message is discarded.

(5) In conjunction with t, the current time T_1 at x_1, a maximum age limit $\mathbf{a}$, and synchronization error Δt, x_1 discards replies that arrive when $\mathbf{a} \leq (T_1 - t)$.

(6) Router x_1 maintains a table containing destinations $\mathbf{d}_r$ and matching progress limits $\mathbf{p}_r$ for which it has requested flooding replies. It discards these entries after an amount of time $> \mathbf{a}$ has elapsed. Any reply it receives containing a source route $\{x_1, x_2, \ldots, x_{n-1}, x_n \mid d\}$ for which $\mathbf{d}$ does not match one of the $\mathbf{d}_r$ is discarded.

(7) From the definition of limited flooding, we assume that router x_{n-1} determined that its immediate neighbor x_n made progress toward $\mathbf{d}$. This is verified by x_1 when it compares the location of x_n against the matching $\mathbf{d}_r$ and progress limit $\mathbf{p}_r$. If progress was not made, the message is discarded.

This demonstrates that the only flooding replies accepted by a router x_1 are those that it has requested, that are valid, and that are no older than the age-limit $\mathbf{a}$.

One of the two principal drawbacks to neighbor validation was that the sets N_{2i} contained no information concerning the quality of any of the links in a source route. The sets can only be used to validate topology. However, properties (1) and (2) ensure that the message containing the source route must itself use that route. Although the quality of the individual links along the source route is unknown, the fact that x_1 accepts a source route implies that the route $\{x_1, x_2, \ldots, x_{n-1}\}$ was functioning within age-limit $\mathbf{a}$.

Providing false information during the flooding enquiry procedure does not create a damaging avenue for attack. If the source address, destination address, or progress limit fields have been altered during the flooding enquiry, this can be detected by the initiating router x_1 when the response arrives. Assume that some router lies during the flooding enquiry and that the initial portion of the source route $\{x_1, x_2, \dots, x_{n-2}\}$ is false when it arrives at x_{n-1}. Since that same route is validated upon return to x_1, either a valid source route is returned to x_1, or the invalid route will be detected and the response discarded.

The techniques developed here cannot be generalized to guard the routing data against two or more simultaneous attacks. If collusion between two routers is allowed, a source route of length greater than two hops cannot be validated using the sets N_{2i}. If x_{n-1} divulges the session key K_S to any router $\{x_2, x_3, \dots, x_{n-2}\}$ along the source route, then invalid source routes can be presented to x_1 that cannot be detected.

6.2 Achieving Attack Resistance using a Validation Authority

A regional authority must possess the sets N_{1i} or the equivalent for each router i=1, ... , n−1 in the source route $\{x_1, x_2, \dots, x_{n-1}, x_n\}$ if it is to validate the addresses and topology along that route. The simultaneous requirement of data integrity requires the use of encryption. It seems natural to combine both validation and key distribution functions within an authentication server. Each authentication server would contain the topology, addresses, and private keys of the routers i in its region.

$$\begin{aligned} x_{n-1} \rightarrow \mathrm{AS}: &\quad x_{n-1}, \{x_1, r\}^{K x_{n-1}} \\ \mathrm{AS} \rightarrow x_1: &\quad \mathrm{AS}, \{r, t\}^{K x_1} \end{aligned}$$

Assume that router x_{n-1} is responding positively to a flooding enquiry from x_1. Router x_{n-1} wishes to send a response to x_1 and to preserve the data integrity of its response. AS validates the addresses and topology along the source route contained in the response. This corresponds to steps (1) and (2) discussed in the previous section. AS then extracts the address x_1 of the router that initiated the flooding response and constructs a reply, which contains the initial flooding response r and a time-stamp t. The reply is then enciphered with x_1's private key and sent by AS to x_1.

Upon receipt x_1 applies steps (3), (5), (6), and (7). At the conclusion of these procedures x_1 can make the same claim made in the previous section. Note that the response from AS would usually travel directly toward x_1 and not to x_{n-1} and then back to x_1 via the reverse source route, as it did under the neighbor-validation scheme above. Router x_1 *cannot* infer that the source route was recently functioning, as it can under neighbor validation. However, since the authentication servers are assumed to be both reliable and secure, the integrity of any validated flooding response that arrives at x_1 is assured. The only attacks that can be made upon the routing data are by x_{n-1} itself or by some router along the source route occur *before* the route is validated.

An obvious drawback of using an authentication server both to store private keys and to validate the source route is that the authentication server itself becomes a potential weak point. A break in the security surrounding an authentication server allows an attacker to freely lie about routes. Under neighbor validation the authentication server only supplies keys and never sees the source route itself. However, Cartesian routers accept routing updates only in response to requests that they themselves have recently made. This characteristic limits the effect that a malicious server can have on the network in its vicinity.

It was shown earlier how rapidly the required storage within an authentication server grows with the maximum allowed length l of a source route. This problem becomes more severe when the sets N_{1l} or their equivalent must be stored. It is possible to subdivide the validation of source routes of length greater than l in a manner similar to that used under neighbor validation.

Assume that an authentication server **AS** is asked to validate a source route $\{x_1, x_2, \ldots, x_{n-1}, x_n\}$ of length greater than l. By itself it can validate the last portion of that route $\{x_m, \ldots, x_n\}$, but it does not possess sufficient knowledge to validate the remainder of the route $\{x_1, x_2, \ldots, x_m\}$. For each of its neighboring servers, **AS** builds a packet that requests the validation of the remainder of the route. It then enciphers the packet with each neighbor's private key and transmits these packets to them using the source routes it maintains.

$$AS \rightarrow AS'\text{:}\quad AS,\ \{x_1, \cdots, x_m\}^{K_{AS'}}$$

$$AS' \rightarrow AS\text{:}\quad AS',\ \{x_1, K_{x_1}\}^{K_{AS}}$$

Each authentication server that receives such a packet decrypts it using its private key. If the neighboring server does not contain information concerning the route segment $\{x_1, x_2, \ldots, x_m\}$ it discards the packet. If it does contain the requested information it builds a reply packet enciphered with **AS**'s private key that contains x_1's private key, and transmits this back to **AS**. This allows **AS** to complete its task. Within limits, this procedure can be generalized to validate longer source routes. Each server would validate that portion of the source route that it could, and would then forward a request to validate the remainder to its neighboring servers. If the route could be entirely validated, a response would eventually return to **AS** via those servers involved in the validation of some portion of the route. However, considerable additional overhead and time is required.

6.3 Limiting the Frequency of Authentication Server Access

If authentication servers are accessed too often they become a choke point, limit network traffic, and raise the cost of maintaining data integrity to an impractical level. The access rate to an authentication server is directly related to the frequency with which routers respond positively to limited flooding enquiries. How often does that occur? First, the issuance of a flooding enquiry by a router is usually a rare event, except at that router's initialization. Second, assume that router x_1 issues a flooding enquiry, receives as a response a source route terminating at router x_n, and then enters x_n's location and the source route to it into its routing table. This implies that x_1 need not again issue a flooding enquiry for any member of the set of destinations for which x_n makes progress. This remains true as long as x_1 is not reinitialized and the source route between x_1 and x_n remains usable.

Although the issuance of a flooding enquiry by any one router is relatively rare, a single enquiry may provoke many responses, thus requiring an equivalent number of requests to an authentication server. The number of routers that respond to a flooding enquiry is limited strictly by topology. However, the number of separate requests to an authentication server made by any one router in response to a specific enquiry can be held to one.

Assume that x_1 issues a limited flooding enquiry to which x_{n-1} responds after communicating with AS. Depending upon topology, x_{n-1} may receive more than one copy of the enquiry, each copy arriving at x_{n-1} by a different route. Additionally, x_{n-1} may have more than one neighbor x_n that makes progress. If neighbor validation is used, then for those subsequent enquiries x_{n-1} need not again communicate with AS, since it already has done so once and the [source, destination] key and other data retrieved from AS by x_{n-1} does not change until a different enquiry is issued by x_1.

6.4 Routing Table Modifications

Some extensions have been made to Cartesian routing in the interest of efficiency. Two of those involve the addition or deletion of routing table entries as the result of supposedly valid information received from other routers. The effects of an attack on these practices must be discussed.

6.4.1 Implicit Routing Table Additions

In the description of Cartesian routing, any router x_i along the source route taken by a flooding reply may by implication enter the route $\{x_i, \dots, x_n\}$ into its routing table. When using neighbor validation, the entire source route $\{x_1, x_2, \dots, x_{n-1}, x_n \mid d\}$ is not validated until x_1 checks its integrity, so such entries should not be made until then. Upstream routers must set aside that route and wait until it is actually used by x_1 before making these entries. If one assumes that there exists at most one attacker, then when a router sees a source route that matches a flooding reply it has recently seen, it may assume that the source route has been validated and enter it into its database.

6.4.2 Implicit Routing Table Deletions

Assume that x_i discovers its link to x_{i+1} in a source route $\{x_1, x_2, \dots, x_{n-1}, x_n\}$ no longer responds. It then transmits a *source route outage* packet using the reverse route $\{x_i, x_{i-1}, \dots, x_2, x_1\}$. This informs those routers that the source route is no longer usable. If x_i is malicious, it can simulate an outage merely by discarding selected packets. There is little to gain from confirming the validity of an outage report.

Router x_1 should now choose another source route that satisfies progress criteria for those destinations that would have used the now inoperative source route. This is achieved either by initiating a limited flooding procedure or by consulting a table of previous limited flooding responses. In both cases, those routes that avoid x_i and x_{i+1} should be preferred.

6.5 Indirect Attacks on Data Traffic

The indirect attacks that can occur during limited flooding have been discussed at length, and provisions for either preventing them or detecting them when necessary have been developed. However, there remains a possibility of attack during data packet transmission.

The following indirect attacks can occur within a data packet during transmission:

(1) Alteration of the source address.

(2) Alteration of the destination address.

(3) Alteration of the data.

(4) Alteration of the progress limit field.

(5) Alteration of the source route, if any is used.

Classes 1, 2, and 3 can break connections or prevent packet delivery. As discussed earlier, unless this pattern of attack is consistent, it may be undetectable. However, neither of these attacks seriously damage the network.

However, by altering a source route an attacker creates the potential for widespread network damage. If a routing loop is created, those routers in the loop will handle a smaller total of useful traffic and possibly become overloaded. If an attacker creates stable routing loops and scatters them throughout a network, the network will be severely damaged. That form of attack can be greatly restricted by means of a *time-to-live* (TTL) counter similar to that in the IP packet header [RFC 791]. The TTL counter is set by the source router in the header of each data packet to a value larger than the maximum credible hop count (or elapsed time) from source to destination. It is decremented by each router that forwards a packet. When the TTL field in a data packet reaches zero, that packet is discarded.

If a TTL field is used, and assuming that at least one router in the loop is operating properly, then a loop cannot be sustained indefinitely unless the TTL field is periodically reset upward; otherwise the counter would eventually reach zero. Therefore, to sustain a loop, at least one attacker must be in that loop to reset the TTL field. This is essentially an indirect attack, characterized by the generation of artificial traffic, and cannot be prevented. If routers in a loop do become overloaded, congestion-control provides some relief.

It is desirable to avoid using a TTL field if possible. Consider a source route $\{x_1, x_2, \dots, x_{n-1}, x_n\}$. For a loop to occur, at least two of the of x_i in the source route must be identical. Assume that each router applies the procedure in step (2) of Section 6.1.1 to each incoming packet that has a source route. Any loop in the source route that exists when that procedure is applied to a packet will be detected and that packet will be discarded; thus the TTL field is not necessary.

6.6 Congestion Control

One category of attack that cannot be prevented is the generation of artificial network traffic. A router can fabricate and transmit packets, which could overload the network in its vicinity. An overloaded router is forced either to discard packets or to shed load by rerouting some of its traffic. If an overloaded router can use a congestion-control mechanism that causes an immediate neighbor to reroute a portion of its traffic, then the damage caused by the generation of artificial traffic can be controlled to some extent.

Cartesian routing provides a congestion-control mechanism for individual routers. A router that is becoming congested can initiate a *congestion-control* flooding request in an attempt to reroute some traffic that uses it as an intermediary. Assume that routers **r** and **y** are neighbors and that **r** chooses as the destination in its flooding request a location for which **y** now forwards traffic to **r**. If **r** issues that flooding request only to **y**, then any legal flooding replies will return via **y**. These replies will create alternate paths for a set of destinations for which **y** currently forwards traffic via **r**. The flooding request implies that **r** would reroute that traffic back via **y**, so **y** recognizes that **r** is a cul-de-sac for that set and reroutes that traffic via one of the routes supplied by the flooding responses [Finn-87]. Furthermore, **y** is forbidden from forwarding any of the traffic of that set to **r** until it has been notified by **r** that the congestion has eased.

If **r** receives traffic from **y** for destinations within the forbidden set, then it may infer that **y** is disobeying congestion-control procedures. Once such a failure is detected, the discovering router can assume that the router which failed to obey the directive is faulty. However, **r** may not receive any replies to its congestion-control flooding request. Responding to congestion-control flooding requests is optional. Rather than responding to a request, a congested router ignores it. This avoids the problem of a congested router rerouting some of its traffic into another congested router. Hence, if no responses to **r**'s flooding request return via **y**, then **r** can only safely assume that the routes which make progress via **y** are currently congested.

Indirect attacks made by abusing congestion-control flooding or illegally ignoring flooding requests will not generally affect the network outside the immediate vicinity of the attack. Discarding flooding requests without being congested only harms the network to the extent that traffic is needlessly routed around it. This becomes serious only if that router is on a critical path, or if the network in its region is already heavily congested. The effects on the network are similar when a router needlessly issues congestion-control flooding requests, which also reroute a portion of its traffic. Abuse of this facility could prevent the network from using an attacking router as an intermediary.

A router is unable to know whether a neighbor obeys the congestion-control flooding procedure. In the above example, **y** could attack **r** by generating artificial traffic that congests **r** and then discard any congestion-control flooding packets sent by **r**. In this situation **r** may issue other congestion-control flooding requests to immediate neighbors other than **y**. If after that **r** still remains congested, it will be necessary for it to discard packets to relieve the congestion, or to sever its connection to **y**. This form of indirect attack will damage the network outside the immediate vicinity of **y**, since those packets discarded would normally include traffic from remote sites. However, the attack's effect is limited to traffic that **y** creates or that is flowing through **y**'s immediate neighbors.

6.7 Host Mobility

Cartesian routing adapts easily to the addition of host mobility. A mobile host **m** is assumed to be in two-way communication with at least one relay $\mathbf{r}_i$, as portrayed in Figure 7a. Relays are routers that have the additional capability of communicating with mobile hosts. Mobile hosts make and break associations with relays as they move. In each acknowledgment (or reply) packet, a mobile host **m** includes the addresses of the set $\{\mathbf{r}_i\}$ with which it is currently associating. This allows the other end of a connection **h** to redirect its traffic toward the appropriate relays as **m** moves. To what extent does mobility affect attack resistance in a Cartesian network?

Any router could synthesize acknowledgment packets for a mobile host. It is possible for a malicious router to masquerade as a member of $\{\mathbf{r}_i\}$ and then potentially to redefine the set $\{\mathbf{r}_i\}$ for a mobile host. The result is to redirect traffic for **m** to the wrong relay. In Figure 7b, if $\mathbf{r}_3$ masquerades as a relay for **m**, traffic directed to **m** from **h** will not reach $\mathbf{r}_1$ and so will not reach **m**. If $\mathbf{r}_3$ can also successfully masquerade as **m**, then **h** may not detect this.

However, a number of preconditions must be met for this to occur.

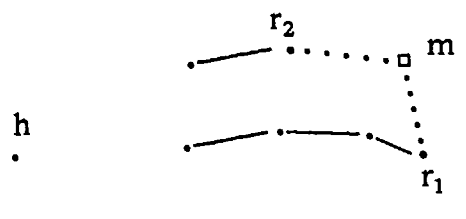


Figure 7a.

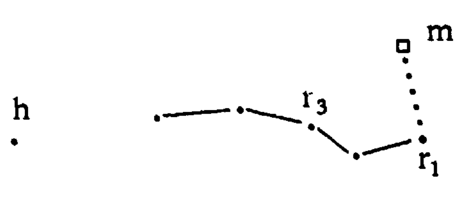


Figure 7b.

(1) Either a connection to **m** must be already open, or someone must be attempting to open a connection to it.

(2) The host **h** must receive acknowledgments from $\mathbf{r}_3$, implying that $\mathbf{r}_3$ knows **h**'s address.

(3) acknowledgments contain sequence numbers, $\mathbf{r}_3$ must supply the correct numbers. If not, then either its acknowledgment will be discarded by **h**, or the connection to **m** will be broken.

In general, for $\mathbf{r}_3$ to steal **m**'s traffic with any reasonable chance of success, it must somehow monitor either **h**'s or **m**'s traffic. This can be further guarded against by including in the neighbor validation sets N_{2i} information declaring whether or not neighboring routers are relays. This would allow $\mathbf{r}_3$'s neighbors to immediately detect its masquerading as a relay.

For some host **h** to send any packet to a mobile host **m**, at least one of the set of **m**'s current associations $\{\mathbf{r}_i\}$ must be known to **h**. When a connection is opened, either some trusted server is consulted or **h** must determine $\{\mathbf{r}_i\}$ by enquiry. If a limited search is performed in the region of the network (presumably, but not necessarily, near **m**) to determine $\{\mathbf{r}_i\}$, then any relay within that region could lie about an association with **m** and could as a result masquerade as **m**. Of course, end-to-end encryption techniques could prevent this from being successful for an extended period.

It is difficult to correspond the type of damage caused by this indirect attack with any previously discussed category of direct attack. Although the damage caused by this indirect attack is not limited to the immediate vicinity of the point of attack, the success of the attack requires that the attacking relay await a specific opportunity that allows it to make the attack. A malicious relay cannot use characteristics of the network operating software to attack connections to mobile hosts whose traffic it cannot monitor. It seems fair to conclude that the network as a whole remains attack resistant even if this form of attack is not detectable.

6.8 Limiting Attack by Broadcast or Excessive Transmission

The cost to an internetwork of transmitting a broadcast packet is usually much higher than that of transmitting a normal point-to-point packet. The limited flooding mechanism within Cartesian routing uses broadcasting to accomplish its objective, and that is therefore a potential weak point.

6.8.1 Limited Flooding

The limited flooding procedure uses high-priority broadcast distribution to find a source route segment that makes progress toward a location. The number of routers within **f** hops of the router initiating a limited flooding procedure is bound from above by $C(C-1)^{f-1}$. The cost to the network of each procedure activation is potentially high. If a router were to continually issue flooding requests, this exponential cost could effectively halt the network in that router's vicinity. This is aggravated if no limit on f is enforced or if that limit approaches the network's diameter. Under those conditions, excessive broadcasting could effectively halt the entire network.

A router does not require the use of limited flooding unless a neighboring link becomes inoperative, or the router wishes to forward a packet with a destination for which its routing table cannot make progress. Both are normally infrequent events. Only at router initialization would one expect the need for more than one flooding request to be in progress simultaneously. It is reasonable to restrict limited flooding requests issued by any single router to no more than one outstanding request at any time. However, self-enforcement procedures are insufficient to protect the network.

Instead, consider letting the immediate neighbors of the issuing router limit the frequency of requests. Each router i can maintain a time-stamp that notes the last time it received a flooding request from each of N_{1i}. Given a reasonable estimate for the typical round-trip time of a flooding request t_r, any flooding request received from a neighbor, that falls within t_r of the last request issued by that neighbor, is discarded. If a router needs to issue more than one request simultaneously, it can serialize them and separate them by an interval greater than t_r.

That is an inadequate solution, because a malicious neighbor could fabricate flooding requests from N_{2i} or yet farther distant. A method that avoids that pitfall is for each router to filter all incoming flooding requests in the following manner. The filter consists of an associative memory table and a first-in-first-out (FIFO) packet queue. The length of both is determined by a reasonable limit on the number of simultaneous flooding requests. When a flooding packet is received, the following actions take place:

1. The source address of the flooding packet is matched against the associative table. If a match is found or the table is full, then the packet is discarded.

2. The packet is stored into the FIFO queue to await processing.

A malicious router cannot overload its region of the network with flooding packets. Any attempt to do so will result in the great majority of the flooding packets being discarded by its neighbors, because either their filter table lengths are exceeded or source address matches occur in their filters. Because the filter can be implemented in hardware, it need not retard normal packet processing. This technique guards against simultaneous attacks made by two or more non-isolated routers.

6.8.2 User-Initiated Broadcasting

A similar risk is presented to the network by user-issued broadcast packets (as opposed to router-issued broadcast packets). Cartesian routing allows *area-limited* broadcasting and enquiries. Given a location (x, y) and a radius Δd, a packet can be broadcast to all routers within the delimited circle. Similarly, a enquiry can be made from all routers within the delimited area. Preventing network damage from excessive user broadcasts can be achieved by a filtration mechanism similar to the one just mentioned.

The gateway through which a host enters the network should have a filter similar to that just discussed. It should filter the broadcast packets initiated by its associated host population. An additional filtration mechanism could be triggered by a router when it receives a broadcast packet and is within the region delimited by Δd. It is within that region where the cost of the broadcasting becomes most noticeable. The maximum radius Δd must also be limited by the network administrator. For a network of metropolitan or national scale, a user should be allowed to broadcast to only a limited portion of the network at any one time. Developing a realistic limit on Δd will require operating experience.

6.8.3 Excessive Generation of Traffic

The filtration technique that protects the network from excessive generation of multicast packets should not be applied to attacks that are characterized by the excessive generation of normal data packets. Under typical conditions, the number of different packets and addresses simultaneously in transit through a router can make that filtration impractical. If incoming traffic load becomes excessive in a Cartesian network, congestion-control mechanisms provide some relief.

6.9 Modifying the Definition of Progress

The Cartesian routing procedure prevents the occurrence of routing loops by requiring all source route segments to make progress toward the destination. A complete path from source to destination is built up by concatentating these segments. In general, there is a set of paths that meet the progress criteria between source and destination. This distinguishes Cartesian routing from other routing procedures, such as those based upon shortest-path routing, in which only one path is shortest at a given time. Since a set of paths is available under the Cartesian procedure, if more than a single path makes progress toward a destination, traffic for that destination can be transmitted over each path. Other conditions can be added to the definition of progress to further refine path selection for differing traffic categories. This allows a certain amount of specialized path selection based upon the type of service desired. In these cases, packets must be marked with their category so that the correct subset of paths is selected. What affect will this have upon a network's attack resistance?

Assume that the internetwork handles both normal and highly sensitive traffic, and that a sizeable portion of the network routers maintain links with differing levels of security. An example of this would be links with enciphered (as opposed to unenciphered) communications channels. The concept of packet progress for sensitive traffic could be redefined to be a source route segment that not only got closer to the destination, but that also used *only* secure channels. For sensitive traffic the flooding algorithm would restrict its searching to secure links. The returned source routes would then be composed entirely of secure links. The routing algorithm would facilitate what has been called *red-black* separation.

The paths to the destination taken by sensitive traffic must not use any non-secured links; so in general, sensitive and non-sensitive traffic must take different paths to the same destination. This is true even if the source of both classes of traffic is a single host. If both paths exist within the flooding search limit, the modification to the definition of progress allows the Cartesian procedure to develop and simultaneously use both paths. Further, if more than a single secure path makes progress, then sensitive traffic for the corresponding destination may be randomly apportioned between the paths. This would make traffic analysis considerably more difficult. It would be desirable to abort any sensitive connection if at any time no fully secured path existed between source and destination.

The source route segment developed by limited flooding can be validated as one that contains only secured links. To achieve this, the information distributed concerning links in the sets N_{2i} must contain the necessary additional data to validate the route. If only secured links are to be used, then during a flooding response each router must validate not only that the link and routers at either end exist, but also that the links are secure. By repeated application the entire route from source to destination is determined to be secure, or the connection is aborted. An individual router may attack the network by transmitting secure data over a non-secure path. This may be impossible to detect.

Assume that the sets N_{2i} contain maximum link capacity in addition to address and topology data. Consider a modification to limited flooding that requests a least upper bound of static link capacity. The definition of progress could be augmented to include a restriction that no source route segment may have less than a stated amount of maximum capacity. A variant of the neighbor validation procedure can ensure that the entire source route has at least the requested static capacity.

This modification provides a weak form of flow control. Paths with limited capacity could be selected for low bandwidth, transaction-oriented traffic, such as TELNET. Conversely, if high throughput is desired, this would allow preference to be given to paths composed of satellite links rather than terrestrial links. However, if no high-bandwidth paths were available, a router could select lower bandwidth paths as a less desirable alternative. Numerous variations are possible.

Notice that the validation techniques discussed in this paper allow efficient validation only of static information, such as addresses, installed topology, and link configuration data. Dynamic data, such as the currently available link capacity, cannot be efficiently validated. The flooding procedure can easily be augmented to search for routes with a minimum available capacity, but that cannot easily be validated. However, neighbor validation will detect any single attack in which a link is claimed to have greater than its maximum capacity.

6.9.1 Router Avoidance

Consider the situation in which it is desirable to avoid using a particular router r except when necessary, perhaps because it is known to be less reliable. Assume that one of r's neighboring routers n_i, an element of N_{1r}, has discovered this. Its definition of progress can now be augmented to avoid any path that uses a link to r. Router n_i examines its routing table for any source route segments $\{x_1, \dots, n_i, r, \dots\}$ and then transmits source route outage packets for them. This ensures that r will no longer be used as an intermediary by n_i. Traffic to r through n_i will be restricted to traffic that is destined for r. This mechanism could be used to provide a weak form of flow control, which offloads the link between n_i and r.

6.9.2 Forced Reconfiguration

The technique of router avoidance can be extended in conjunction with authentication servers to provide an attack-resistant method that forcibly deconfigures a particular router r from the network. If r is known to be malicious or very unreliable, it is desirable that it be deconfigured from the network until control over it is regained or until it is repaired. If r is malicious, administrators cannot assume that it would obey any 'shutdown' messages that it might receive from network administration; nor can they assume that it would reliably communicate such messages to its neighbors.

To remotely deconfigure r from the network it is necessary to inform r's immediate neighbors to cease communication with r. This requires a reliable communication route to r's neighbors from a network administrative authority. The straightforward solution is to use the regional authentication servers. The route can then be constructed in the following manner.

Assume that a regional authentication server AS wishes to reliably communicate with a router r's immediate neighbors N_{1r}. AS issues a set of unique limited flooding requests, each specifying one of N_{1r} as the destination with zero in the progress limit field. Each request returns a set of source routes from AS to an element of N_{1r}. However, router r must be avoided by all such paths because it is assumed unreliable. By removing from those sets any source route that contains r, a reliable path to each element of N_{1r} is constructed. The set selection criteria can be made more severe, such as forbidding any path to contain a router from a set $\{r_i\}$. However, doing so increases the probability that no acceptable paths remain.

Implicit in this method is the assumption that the network is sufficiently well connected that each of AS's flooding requests returns a non-empty set of source routes. Since forcible deconfiguration should occur very infrequently, the value of f can be set sufficiently high to ensure non-empty return sets in all but the most poorly connected of network topologies. However, it is still possible that the removal of routes containing r would result in an empty set. For example, r could be on a cut path that partitions the network.

Section 1.3.3 discusses single attacks that involve the discarding of messages. If a router consistently discards traffic for a particular connection, the only evidence for the attack may be a severed connection. If it is known that both source and destination are properly operating and that no network error message of the form "connection dropped due to no route" has been received, one possible cause might be an attacking router that is discarding messages. The attack would have been made somewhere along the route chosen by the network.

Under these circumstances, one could postulate a trial-and-error network reconfiguration program designed to detect and eliminate the malicious router or link. An administrative center could perform the following actions:

(1) Selectively deconfigure from the network a router and its links, choosing the router from along the presumed connection path.

(2) Inject at the source router traffic with the same profile as that which originally provoked the attack.

(3) Determine whether or not traffic is flowing over the connection. If it is flowing, then presumably the malicious router has been removed from the network. If it is not flowing, then reconfigure that router and return to step (1), choosing a new router.

Steps 1 through 3 would be repeated until the offender has been found or until the routers along the presumed path have all been chosen and tested.

6.10 Random Use of Multiple Paths to Destinations

It will always be possible for any link or router to discard traffic, to alter the traffic that passes through it, or to fabricate traffic. Given the vulnerability of links and routers, this cannot be prevented, and the effects of an attack on a connection are felt outside the immediate vicinity of the attack. This form of indirect attack is in a sense an irreducible minimum. A routing protocol that already resists other forms of indirect attack and that could also resist this form of attack would truly meet the definition of attack resistance.

Assume that an attack occurs in a link or router along the path between source and destination. Ideally, if the attack became known to the attacker's neighboring routers, then traffic could be routed around the attacker. However, it was pointed out earlier that such attacks may be undetectable or impractical to detect. If such is the case, the continuity of a connection may only be ensured by a policy that uses retransmission in conjunction with multiple paths over which traffic for the connection flows. To guarantee this, each path must share no routers or links in common. In most cases that would be impractical, since connection endpoints often only access the network via a single router or gateway, and for datagram procedures a router may not possess knowledge beyond the next hop on a packet's path.

Random Router Selection Between Multiple Routes

Consider a statistical approach that closely approximates the ideal solution. Section 4.10 of this report mentions that routing procedures, such as Cartesian routing, which can simultaneously develop and use more than one path between source and destination, possess a distinct attack-resistant advantage. Since limited flooding returns a set of source route segments that make progress toward a set of destinations, a Cartesian routing procedure in many cases may simultaneously maintain in its routing tables optional routes that it can use to make progress toward a destination. This approach could also be used in conjunction with virtual circuit routing.

To successfully forward packets toward any destination, a router need only maintain a single source route segment in its routing table that allows it to make progress. Assume that each router retains in its tables all the acceptable flooding replies that it receives to its flooding requests. Assume also that if a router has an immediate neighbor that makes progress for a class of destinations, then that router issues a limited flooding request to obtain addition routes that make progress. Finally, assume that connection-level protocols implement end-to-end retransmission and are able to detect duplicate packets.

The routing path selection policy is now modified. Rather than selecting from the routing table the single source route segment that is thought to be the most efficient, each router when possible randomly chooses a route segment from the set of segments that make progress. This policy would cause the network to route connection traffic over a variety of paths for most connections. However, this policy would tend to decrease network efficiency, and under congested conditions it may be difficult for a router to maintain the necessary sets of routes.

If this policy is followed, an attacker may intercept some traffic for a connection, but it is unlikely to see more than a fraction of it unless the attack occurs on a critical path for the connection. It then becomes much less likely that an attacker can break a connection or prevent one from occurring. End-to-end retransmission would ensure that both acknowledgments and duplicates, when transmitted, would take a variety of paths. Without knowledge of correct sequence numbers, it becomes difficult for an attacker to cause a connection to reset or to fabricate traffic acceptable to the source or destination. Although an attacker could prevent a connection from opening if it merely discarded the initialization packets, repeated attempts to open the connection from the source would be likely to succeed because copies of the initialization packets would probably take different routes.

Further refinements are possible. Consider a routing table and a set of routes that make progress toward a destination. Although each alternate path within the set starts from the router making the choice, it is desirable that elements of the set share as few common members as possible. Ideally, one desires a set of at least two paths that share no members except for the starting router. The possibility of sets of paths meeting that criterion when passing through a network region of low connectivity is unlikely.

Finally, there is the question of packet duplication. When should this policy be used? When used, should end-to-end retransmission be relied upon to ensure delivery of undamaged packets and to maintain a connection, or should duplicates also be generated by the source router so that at least two copies of each packet are transmitted? The answers probably depend upon the relative importance of a connection. The possibility of duplicate transmission by each router that makes a routing choice (not intermediary routers in a source route segment) would probably generate too much overhead. Although it may not be possible to achieve a quantitative answer, the question of how much more reliably connections are maintained by this policy in the presence of attack is an interesting one.

6.11 Pros and Cons

Mechanisms have been developed for Cartesian routing that ensure the validity of source routes discovered via limited flooding. If these mechanisms are employed, no single router or filter on a link can supply invalid routing information that goes undetected. In addition, it was shown that adding link encryption and implementing filtration mechanisms for broadcast packets can greatly restrict the damage that results from most other classes of indirect attack. Finally, Cartesian routing can be modified to closely approach the ideal of attack resistance through the use of a statistical policy of route selection.

These mechanisms are practical, but there are costs. Authentication servers must be added to the network, neighbor-validity sets must be maintained, and normally an additional two-packet authentication exchange must occur prior to the transmission of a flooding response, thus lengthening the response time to a flooding enquiry. The cost of authentication is not too high as long as the occurrence of limited flooding remains infrequent. The link encryption and filtration mechanisms can be easily implemented in hardware so that any performance degradation resulting from their use is small. The statistical route selection policy does complicate router software and decrease overall network efficiency. Its effectiveness also decreases as a network becomes congested.

7 Conclusion

Practically speaking, there is no way to completely protect a system against hardware failure or human intervention. Hardware is not completely reliable, checksumming cannot guarantee that all data corruption will be detected, and ultimately some people concerned with system operation must be trusted. The design of a network's operating software should be defensive and attack resistant, so that a failure or malicious attack at any one point cannot seriously affect network-wide operation.

A routing procedure that widely disseminates routing information among routers raises the possibility that a malicious attack or failure at any single router could severely disrupt the network. The shortest-path routing procedures examined in this report exhibit this behavior; therefore, their use is inappropriate in situations where network operation fulfills vital needs. It is not crying 'wolf' to suggest that this is a disaster waiting to happen; it has already happened once unintentionally. If a malicious attack occurs, the probability of serious network disruption - or even failure - is high.

To improve the attack resistance of a network the following actions, listed in rough order of increasing importance, should be taken:

(1) Ensure sufficient richness in topology that the smallest network routing cut set is at least two.

(2) Do not multiplex separate links into the same physical communications medium.

(3) Employ link encryption, with unique private keys stored in each router for its links.

(4) Employ a routing procedure that strongly limits the influence any router can have on the routes chosen by other routers. If possible choose one that also assures the integrity and validity of distributed routing information.

(5) Employ a routing procedure that allows routers to simultaneously maintain a set of alternative paths between [source, destination] pairs. The sets of routers on those paths should be as nearly disjoint as possible. For each packet, follow a policy of randomly selecting a path from among the set of alternative paths.

(6) Force routers to limit their own and their neighbor's use of high-priority or multicast transmission. A router should impose similar restrictions on its population of hosts.

(7) Do not use any network operating software that incorporates mutual restrictions in host or router behavior which, if violated, could severely damage the network.

(8) Carefully protect any remote network maintenance procedures.

The Cartesian routing procedure exhibits the attributes desired in an attack-resistant routing procedure. It can serve as the kernel of a suite of operating software that allows a network to become attack resistant. In the future, the builders of any widespread network should seriously consider the question of attack resistance before designing a routing procedure or any other piece of network operating software.

Appendix I

Synopsis of the ARPANET Updating Procedure

I.1 Update Age Field

An age field is associated with each routing update. Each IMP (ARPANET router) decrements this field while it is in possession of the update prior to transmission. The intention is to ensure that old updates do not circulate within the network beyond their reasonably useful lifetime. Each IMP maintains a table of most recent age-values associated with the updates it has received from each IMP in the network. The following restrictions are associated with the age field:

- When an IMP originates an update packet, the age field in the packet is set to a maximum value of eight.
- When an update **u** from IMP *j* is accepted by an IMP (see below), the age field of the update is copied into the age-vector entry associated with IMP *j*.
- Each IMP possesses a slowly ticking clock. At each tick (approximately 8 seconds) all the age-vector values are decremented by one, unless the value is zero. These clocks are not globally synchronized.
- When an update from IMP *j* is re-created by another IMP from its internal tables for possible retransmission, the packet's age value is copied from the entry associated with *j* in the age-vector. If the age-value would be zero, the retransmission is canceled.

I.2 Update Serial Number

A serial number field is placed into each update. It is used by the receiver of an update, in conjunction with the originating IMP's identity, to determine whether or not this update is more recent than the previously received update from the same IMP. Each IMP maintains a vector of most recently seen serial numbers from accepted updates for each IMP.

Each IMP maintains a 6-bit serial number for updates that it originates. When an IMP originates an update it increments this serial number by 1 (modulo 64). In comparing two serial numbers n and m, if $n=m$ then the updates are considered to be duplicates. When n is more recent than m, either $n > m$ and $n-m \leq 32$, or $n < m$ and $m-n > 32$. Otherwise, m is more recent than n.

I.3 Accepting an Update

Assume that IMP i has previously accepted update **u** from IMP j. IMP i now receives an update **u'** from j. Now i must determine whether or not **u'** is acceptable. The new update **u'** is judged acceptable if either of the following criteria are met:

- **u** has an age-value of zero (the age-vector entry associated with IMP *j* has reached zero), and **u'** has a non-zero age.
- The serial number in **u'** is more recent than that associated with **u**.

If **u'** does not match either criterion, it is unacceptable and is discarded. If **u'** has an age-value of zero, it is immediately discarded, since this should never occur.

I.4 Transmission and Acknowledgment

Normally, each IMP receives updates from all other IMPs in the network. When IMP *i* receives an acceptable update, the update is stored internally and then retransmitted over all of IMP *i*'s outgoing lines. IMP *i* expects each update that it transmits to a neighbor to be acknowledged. If an update is not acknowledged within a reasonable time, then that update is recreated, a RETRY bit is turned on in the update, and it is retransmitted.

Stated more precisely, let IMP *i* have k input lines. IMP *i* maintains k 2-bit timers for each other IMP in the network (one per IMP per input line). When an IMP receives an update **u'** from IMP *j*, it checks to see if the RETRY bit is ON. If it is, the RETRY bit is turned OFF and the update is echoed back over the input line. Whether or not the RETRY bit was set, *i* now determines if **u** is acceptable (see above).

Assume that **u'** has a serial number identical to that from the previously accepted update **u** from *j*. Then **u'** is unacceptable. but a special action is taken. The timer associated with IMP *j* and appropriate input line is set to zero. If unacceptable, **u'** is always discarded.

If **u'** is acceptable, its age-value, serial number, and delay information are copied into relevant tables. The timer corresponding to IMP *j* and the input line is set to zero. The timers for *j* on the other input lines are set to three. Update **u'** is now flooded over all lines.

Timers for typical land lines are decremented every 25.6 ms. If a timer reaches zero, the update is recreated from IMP *i*'s internal tables and its RETRY bit is turned ON. The update is then transmitted over the line whose timer reached zero. The timer is reset to three. If one of IMP *i*'s lines has been inoperative and now returns to service, then all updates that do not have age zero are recreated from internal tables and are transmitted over that line with the RETRY bit ON.

If an IMP crashes and is reloaded, it must wait 90 seconds before returning to service. Each operating IMP must send an update at least often enough so that its most recent update does not appear too old to any other non-partitioned active IMP in the network. No IMP can originate updates more frequently than 12 per minute. Since propagation time for an update across the ARPANET is -100 ms., when IMP *j* originates an update, all non-partitioned IMPs will have received that update before *j* can legally generate another.

References

[ANSI X3S3.3]	Draft Network Layer Rôuting Architecture ANSI X3S3.3/86-215R2, April 1987.
[Baratz-86]	Baratz, A.E., and Jaffe, J.M. Establishing Virtual Circuits in Large Computer Networks *Computer Networks and ISDN Systems* 12 (1986) pp. 27-37.
[Dijkstra-59]	Dijkstra, E.W. A Note on Two Problems in Connection with Graphs *Numerische Mathematik* 1:269 (1959).
[Dolev-82]	Dolev, D. The Byzantine Generals Strike Again *Journal of Algorithms* 3, 14-30 (1982).
[Finn-87]	Finn, G.G. Routing and Addressing Problems in Large Metropolitan Scale Internetworks USC/Information Sciences Institute ISI/RR-87-180, March 1987.
[Ford-62]	Ford, L.R., and Fulkerson, D.R. Flows in Networks Princeton University Press, 1962.
[Garcia-Molina-82]	Garcia-Molina, H. Elections in a Distributed Computing System *IEEE Transactions on Computers* Vol. C-31, No. 1, January 1982, pp. 48-59.
[Kamoun-76]	Kamoun, F. Design Considerations for Large Computer Communication Networks Ph.D. Thesis Eng-7642. University of California at Los Angeles.
[Kleinrock-77]	Kleinrock, L., and Kamoun, F. Hierarchical Routing for Large Networks Performance Evaluation and Optimization *Computer Networks* 1 (1977) pp. 155-174.
[Lamport-82]	Lamport, L., Shostak, R., and Pease, M. The Byzantine Generals Problem *ACM Transactions on Programming Languages and Systems* Vol. 4, No. 3, July 1982, pp. 382-401.

[Lenoil-87]	Lenoil, R., and Richards, P. Bogus ROOT domain server on ARPAnet. From a message sent to the RISKS message group June 1987.
[McQuillan-78-1]	McQuillan, J.M., Richer, I., and Rosen, E.C. ARPANET Routing Algorithm Improvements First Semiannual Technical Report Bolt Beranek and Newman, Inc. Report No. 3803, April 1978.
[McQuillan-78-2]	McQuillan, J.M., Richer, I., and Rosen, E.C. ARPANET Routing Algorithm Improvements Second Semiannual Technical Report Bolt Beranek and Newman, Inc. Report No. 3940, October 1978.
[McQuillan-80]	McQuillan, J.M., Richer, I., and Rosen, E.C. The New Routing Algorithm for the ARPANET *IEEE Transactions on Communications* Vol. COM-28, No. 5, pp. 711-719, May 1980.
[Mills-84]	Mills, D. RFC 904: Exterior Gateway Protocol Formal Specification University of Delaware, Dept. of Electrical Engineering April 1984.
[Mills-86]	Mills, D. RFC 975: Autonymous Confederations University of Delaware, Dept. of Electrical Engineering February 1986.
[Mills-87]	Mills, D. RFC 1004: A Distributed-Protocol Authentication Scheme April 1987.
[Mockapetris-83]	Mockapetris, P. RFC 882: Domain Names - Concepts and Facilities USC/Information Sciences Institute November 1983.
[Needham-78]	Needham, R.M., and Schroeder, M.D. Using Encryption for Authentication in Large Networks of Computers *Communications of the ACM* Vol. 21, No. 12, pp. 993-999, December 1978.
[Plummer-82]	Plummer, D. C. RFC 826: An Ethernet Address Resolution Protocol Symbolics, Inc. November 1982.

[Postel-87]	Postel, J. Some Communications Irregularities Personal communication.
[RFC 791]	Jon Postel, ed. RFC 791: Internet Protocol DARPA Internet Program Protocol Specification USC/Information Sciences Institute September 1981.
[RFC 823]	Hinden, R., and Sheltzer, A. RFC 823: The DARPA Internet Gateway Bolt Beranek and Newman, Inc. September 1982.
[Rivest-78]	Rivest, R.L., Shamir, A., and Adleman, L. A Method for Obtaining Digital Signatures and Public Key Cryptosystems *Communications of the ACM* Vol. 21, No. 2, pp. 120-126, February 1978.
[Rosen-80]	Rosen, E.C. The Updating Protocol of the ARPANET's New Routing Algorithm *Computer Networks* 4 (1980) pp. 11-19.
[Rosen-81-1]	Rosen, E.C. IEN 189: Issues in Internetting Part 4: Routing Bolt Beranek and Newman, Inc. June 1981.
[Rosen-81-2]	Rosen, E.C. Vulnerabilities of Network Control Protocols: An Example *Computer Communications Review* Vol. 11, No. 3, pp. 10-16, July 1981.
[Rosen-82]	Rosen, E.C. RFC 827: Exterior Gateway Protocol (EGP) Bolt Beranek and Newman, Inc. October 1982.
[Seamonson-84]	Seamonson, L.J., and Rosen, E.C. RFC 888: "Stub" Exterior Gateway Protocol Bolt Beranek and Newman, Inc. January 1984.
[Srikanth-87]	Srikanth, T.K., and Toueg, S. Simulating Authenticated Broadcasts to Derive Simple Fault-Tolerant Algorithms *Distributed Computing* 2 (1987) pp. 80-94.

[Turner-86] Turner, J.S.
The Design of a Broadcast Packet Network
IEEE INFOCOM '86, pp. 667-675.